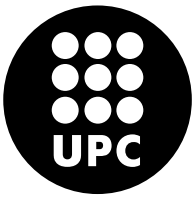


UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONA**TECH**

Facultat d'Informàtica de Barcelona

Master in Innovation and Research in Informatics

High Performance Computing

Master thesis

A Parallel Monte Carlo Algorithm for Solving the Scattering Problem in Plasmonic Nanoparticles

Author: Hector Lopez Menchon

Advisor: Juan Manuel Rius Casals

Tutor: Eduard Ayguade Parra

Defense Date: 25/10/2019

Abstract

The Ulam-Neumann algorithm is a randomized technique employed for solving systems of equations. We propose a blockwise version of this algorithm. The blockwise version may be specially useful for solving systems of equations arising from the electromagnetic scattering problem, that frequently appears in photonics and plasmonics.

The proposed method has good parallel properties and allows the implementation of FFT acceleration techniques.

Keywords: Monte Carlo methods, numerical methods, parallel computing, photonics.

Contents

1 Introduction

1.1 Monte Carlo methods

The birth of modern Monte Carlo algorithms takes place in the late 1940s. However, the idea of using random processes to approximate the outcome of complex computations dates back to many years before. In the 18th century, Georges-Louis Leclerc, Compte de Buffon, proposed a probability problem regarding the position of a randomly dropped needle. Although it was not its original intention, that process can be used to build an estimator of π. In the 1930s, Enrico Fermi used sampling methods to approximate physical quantities in fission problems.

In the 1940s, in the context of the Manhattan Project, the need for solving extremely complex physical problems and the appearance of early computers led to the development of modern Monte Carlo methods. The original idea is usually ascribed to Stanislaw Ulam, although, as we have mentioned, randomized methods were used by Fermi before. The team involved on the development of Monte Carlo methods included Ulam himself, John von Neumann, Nicholas Metropolis and Edward Teller, among others. Today, Monte Carlo methods had become a fundamental tool not only in computational physics, but also in chemistry, biology or economy.

Monte Carlo methods employ stochastic processes to simplify computational problems: when the whole problem is intractablely large, it is possible to randomly pick some samples of the problem, and from them, inferring an approximation of the global solution. They can also be used to quickly compute rough approximations of the problem solution. The sampling is usually determined by a stochastic process, and the probability distribution should be such that the chosen samples represent well enough the whole system.

The Ulam-Neumann method is one of the earliest Monte Carlo methods. It approximately solves systems of linear algebraic equations (SLAE) by taking samples of the Neumann series, which expresses the solution of the problem as a power series. This algorithm is usually employed for extremely large problems or to accelerate some parts of deterministic methods where a rough approximation of the solution of a SLAE is needed. It has gained renewed interest in the Big Data era [1]. In some applications the required accuracy is very low compared with many numerical problems (about 10^{-2} or even 10^{-1}) and the data are extremely large, so the Ulam-Neumann method can be a valuable approach.

1.2 Proposed application

There are other fields where old Monte Carlo methods can be useful to address new computational problems. Nanophotonics, which studies the interaction of light with objects of nanometric size, is one of them. As in all disciplines involving electromagnetic phenomena, numerical methods are employed to solve Maxwell's equation in complex conditions in order to predict the behavior of the system. However, some nanophotonics problems, as metamaterials or colloidal solutions, generate problems that are numerically huge. Besides, the intrinsic variations of physical systems make a unnecessary to solve the numerical problems in a very accurate way: we known in advance that the accuracy of our solution is going to be limited. These two factors make Monte Carlo methods good candidates for approaching these kind of problems.

In this work, we propose a modified version of the Ulam-Neumann algorithm to address a particular physical problem that frequently arises in the context of photonics and nanophotonics. The physics of the problem and the spatial discretization we chose lead to a system of linear equations with very peculiar characteristics. Our algorithm takes maximum advantage of these characteristics. As we will see later, the fact that the system matrix is formed by blocks with Toeplitz structure can be exploited by extending the algorithm to the block level and accelerating multiplications through FFT. We pay special attention to the relation between numerical issues of the algorithm and the physical nature of the problem.

The Ulam-Neumann algorithm is mainly useful on the context of very large systems [2], [3]. Besides, it is embarrassingly parallel [4]. Thus, this algorithm naturally belongs to the realm of parallel computing. This aspect will be highlighted along the following text both by theoretical reasoning and by appealing to other works in the literature. Nevertheless, this is a mainly theoretical work. The numerical examples we will provide are a proof of concept for the theoretical results, but not an example of the algorithm showing its full capabilities. A true large scale parallel implementation of the proposed algorithm remains for future work.

1.3 Structure of this work

Here we give a short overview of the the following chapters of this work. In section 2 we describe the problem we are trying to solve. After a short physical introduction (section 2.1) we introduce the system of linear algebraic equations that emerges from a certain discretization of our problem (section 2.2). This SLAE is the problem we will address through the Ulam-Neumann

algorithm. Also, in section 2.4 we will review some classical fixed point iterative solvers. As we will see later, there is a theoretical connection between these methods, the Neumann series and the Ulam-Neumann Monte Carlo algorithm.

In section 3 we will describe our contribution. Section 3.1 works as a preamble where we describe the well known algorithm for accelerating matrix-vector products were the matrix has a Toeplitz structure. Also, in this section we provide further information about the structure of the system matrix. In this way, section 3.1 complements section 2.2. Next, section 3.2 describes the classical Ulam-Neumann algorithm, paying special attention to its connection with the Taylor series, which is a point that usually remains obscure in the literature. Section 3.3 constitutes the core of this work. Here, we describe our contribution, that consists on extended the algorithm to the block case and studying its convergence by reproducing part of the work in [5] for the block case. Later, in 3.3.4 we show the advantage of using a block level version of the Ulam-Neumann algorithm for this physical problem: block products are efficient because they can be accelerated by FFT. This is a consequence of the discretization we have chosen. Then, in section 3.4 we present some algorithms were the Ulam-Neumann method can be realistically employed.

Finally, in section 4 we provide some numerical examples of different aspects of both the problem and the algorithm. In light of these examples, we discuss in section 4.5 some critical computational aspects related with the blockwise Ulam-Neumann algorith. Among others, we discuss the expected scalability of the algorithm.

2 The problem

2.1 Scattering by multiple particles

In this work, we focus on a particular physical problem: the scattering problem [6] for a cluster of dielectric objects [7], paying special attention to the photonic and plasmonic cases. Basically, we consider a large set of rather simple physical objects that have some known electromagnetic behavior, which is determined by the physical parameter known as permittivity, usually represented as ε. This parameter depends on the material and on the wavelength and, in general, it is complex. The set of objects is irradiated with a certain electromagnetic field, that can range from radio waves to visible light. This applied electric field is called incident field, $\mathbf{E}_{inc}$ or $\mathbf{E}_i$. The set of objects will produce alterations on the electric field. Depending on the permittivity of the material, the field can be reflected, absorbed or, in general, modified. The equation describing the phenomena is $\mathbf{E} = \mathbf{E}_i + \mathbf{E}_s$, where $\mathbf{E}$ is the total electric field, $\mathbf{E}_i$ is the field we are applying and $\mathbf{E}_s$ is the scattered field, it is, the contribution to the total field due to the physical object. Our goal will be to compute the $\mathbf{E}_s$ given a certain $\mathbf{E}_i$ and a certain set of physical objects with a defined permittivity. This is known as the electromagnetic scattering problem. A schematic representation of this is depicted in fig.1. Although the image shows an array formed by a few elements, the systems we are considering will be formed by hundreds or thousands of elements.

This kind of structures can be used to manipulate the electromagnetic field in specific ways. In this case, due to the outstanding properties they can achieve,these structures are known as metamaterials. For example, some metamaterials show collective permittivity values that do not exist in nature, others are designed to allow or block the light flow depending on the frequency or the direction [8].

Although there are metamaterials for all the wavelength in the electromagnetic spectrum, ranging from microwave [9] to visible [10], our main interest is on photonic and nanophotonic metamaterials. They have some characteristics that made Monte Carlo methods specially suitable for them. First of all, they are extremely complex. Since they work in the visible regime, the wavelength would be about some hundreds of nanometers. Thus, the elements of the metamaterial will be in the order of some tens of nanometer. Due to their small size, it is possible to embed thousands of these elements in a small space. The resulting problem is then extremely large, and, as we mentioned in section 1.1, Monte Carlo methods are a good way to deal with the complexity of some problems.

The second reason is that real metamaterials do not exactly correspond

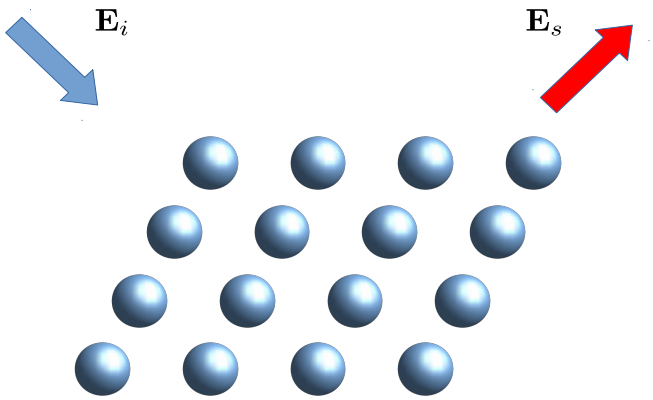


Figure 1: Schematic illustration of electromagnetic scattering by a set of dielectric objects. The incident field $\mathbf{E}_i$ (known) illuminates the array of dielectric particles (known) and, as a result, a scattered field $\mathbf{E}_s$ is generated (to be computed).

to their theoretical design, but use to have flaws. For example, the array depicted in fig.1 is formed by 16 perfect dielectric spheres. In reality, these spheres will not be perfect due to fabrication inaccuracies. Even when it is possible to build structures with a very high degree of precision, it may be prohibitively expensive. Monte Carlo methods, besides a way for reducing the complexity of computations, can be a tool to address the effects of variability. The variability of the physical structure can be modeled with a random variable and included within the structure of the Monte Carlo algorithm. In this work, we will pay special attention to the first of these two reasons.

One particular case within photonics is the plasmonic case. It is known that metal structures of nanometric size interact with light in a very particular way. The electromagnetic optical field induces oscillations in the free electron gas and, at some frequencies, the oscillations of the gas engage in a resonant behavior. Those resonances are known as surface plasmons [11]. Plasmonic effects on metals allow to build nanometric structures capable of manipulating light at subwavelength scale.

From our point of view, the distinguishing feature of a plasmonic material is that the real part of the permittivity is negative. In section 4, we will provide numerical examples of materials with plasmonic behavior.

2.2 Numerical discretization of the problem

Solving the scattering problem is often a challenging task. When the problem is formulating in integral terms, its solution involves solving a Fredholm functional equation. Due to the linearity of the operator, it is possible to discretize this operator into a system of linear equations by applying the Method of Moments [6]. In general, we depart from a functional equation:

$$Lu = f \tag{1}$$

We consider that the solution of the problem can be expanded as a linear combination of functions t_i as

$$u \simeq \sum_{i=1}^{N} u_i t_i \tag{2}$$

where u_i are scalar coefficient. The basis t_i should properly expand the function space where the solution belongs to. In order to find the coefficients u_i, the functional problem is discretized as

$$\sum_{j=1}^{N} u_j \langle w_i, Lt_j \rangle = \langle w_i, f \rangle \tag{3}$$

where w_i is a set of weighting functions.

In the case of electromagnetic scattering problems, the functional problem is

$$L \begin{bmatrix} J \\ M \end{bmatrix} = \begin{bmatrix} E_i \\ H_i \end{bmatrix} \tag{4}$$

where the functions E_i and H_i are the incident electric and magnetic fields, J and M (the unknown functions) are a set of equivalent electric and magnetic currents that appear when the system is irradiated, and L is the so called scattering operator [6]. By knowing the equivalent currents J and M, it is easy to compute the the total scattered field.

The integral representation of the electromagnetic problem can be formulated in many different ways. If the object is a dielectric, one can chose a volume integral equation, where the functions are defined on a set corresponding to the volume of the object, or a surface integral equation, where the functions are defined only on the surface of the body. Of course, the surface integral equation gives rise to a smaller linear system, but it also faces more complicated stability issues. Besides, several linear combinations of scattering operators can be chosen in order to enforce the accuracy of the

system or its convergence properties in different frequency ranges. Also, in order to choose the most suitable test and weight functions, we need to take into account the physical properties of the system.

In our case, we have selected the following volume integral equation [12]:

$$\frac{1}{j\omega\varepsilon_0(\varepsilon(\mathbf{r})-1)}\mathbf{J}(\mathbf{r}) - k^3\int_V \frac{1}{j\omega\varepsilon_0}\overline{\overline{G}}(\mathbf{r},\mathbf{r}')\cdot\mathbf{J}(\mathbf{r}')d^3\mathbf{r}' = \mathbf{E}_{inc}(\mathbf{r}) \tag{5}$$

where ε_0 is the vacuum permittivity, ε is the relative permittivity of the material, ω is the frequency of the incident field, and k is the wavenumber. The dyadic Green's function $\overline{\overline{G}}(\mathbf{r},\mathbf{r}')$ is associated with with the wave equation.

$$\overline{\overline{G}}(\mathbf{r},\mathbf{r}') = (\mathbf{I} + \frac{\nabla\nabla}{k^2})g(|\mathbf{r}-\mathbf{r}'|) \tag{6}$$

$$g(|\mathbf{r}|) = \frac{\exp(ik|\mathbf{r}|)}{4\pi k|\mathbf{r}|} \tag{7}$$

It is sometimes more convenient to write the system in terms of the electric field

$$\mathbf{E}(\mathbf{r}) - k^3\int_V (\varepsilon(\mathbf{r})-1)\overline{\overline{G}}(\mathbf{r},\mathbf{r}')\cdot\mathbf{E}(\mathbf{r}')d^3\mathbf{r}' = \mathbf{E}_{inc}(\mathbf{r}) \tag{8}$$

The discretization is carried out with a simple point collocation technique. The volume is discretized in cubic cells. Then, for a cube j centered in $\mathbf{r}_j$, $t_j = \delta(\mathbf{r}-\mathbf{r}_j)$ and w_j are pulse functions (single point integration). After this discretization, we can write the problem as a system of linear equations [13]

$$\alpha_i\mathbf{E}(\mathbf{r}_i) = \mathbf{E}_i(\mathbf{r}_i) + \frac{k^3V}{4\pi}\sum_{j\neq i}(\varepsilon(\mathbf{r}_j)-1)\mathbf{T}_{ij}\cdot\mathbf{E}(\mathbf{r}_j) \tag{9}$$

where α_i is a phsyical constant, V is the volume of the grid cell, and $\mathbf{T}_{ij}$ is

$$\mathbf{T}_{ij} = \frac{\exp(i\rho_{ij})}{\rho_{ij}^3}(\rho_{ij}^2 + i\rho_{ij} - 1)\mathbf{I} + \frac{\exp(i\rho_{ij})}{\rho_{ij}^3}(-\rho_{ij}^2 - 3i\rho_{ij} - 3)\hat{\mathbf{r}}_{ij}\hat{\mathbf{r}}_{ij} \tag{10}$$

$$\rho_{ij} = k|\mathbf{r}_i - \mathbf{r}_j| \tag{11}$$

$$\hat{\mathbf{r}}_{ij} = \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{12}$$

For the sake of simplicity, the SLAE representation of this problem will be noted simply as

$$Ze = e_i \tag{13}$$

where e represents the total field we are trying to compute, e_i represents the incident field and Z is the so called impedance matrix, that represent the effect of the physical system.

When the meshing is regular, it is, when all the points of the grid are equispaced, the system of the matrix shows a Toeplitz structure that allows to compute matrix products in a very efficient way. We will come back to this point in section 3.

2.3 Spectral properties

The main advantage of this kind of spatial discretization are the good conditioning of the system and the possibilty of obtaining useful a priori information about the spectrum. Samokhin proved [14] that, if the body is homogeneous (i.e., $\varepsilon(\mathbf{r}) = \varepsilon$), then the spectrum of the continous operator is located on the segment $[1, \varepsilon]$. For inhomogeneous bodies, similar localization theorems are available. It is worth to note that the spectrum of the operator does not depend on the geometry of the body, but only on permittivity.

It has been proven [13],[14] that the spectrum of the discretized operator approaches relatively well the spectrum of the functional operator. Then, by taking into account the localization theorems and considering plasmonic permittivities, we can easily conclude that the system is well conditioned. In fact, in our experiments we can see that the segment has an unexpected prolongation after 1 (see figure). Some of the eigenvalue plots in the literature also present this problem, although no clear explanation is given. Nevertheless, we have checked that this unexpected prolongation of the segment does not vary as the discretization mesh is refined. Then, this phenomenon can be taken into account by finding the spectrum of coarsely discretized problem, and assuming that it is going to stay in the same region for finer discretizations.

The properties of volume integral operators contrast with the ones of the surface operators. Surface operators lack this kind of precise spectrum localization theorems, and the matrices tend to be ill conditioned.

2.4 Direct methods for solving the system

In this section, we will describe some classical iterative methods for solving the SLAE associated to the electromagnetic scattering problem. As we will see later, this system are related with the Ulam-Neumann algorithm.

The mathematical properties of the volume integral operator for the electromagnetic scattering made it specially suitable for solving it through a

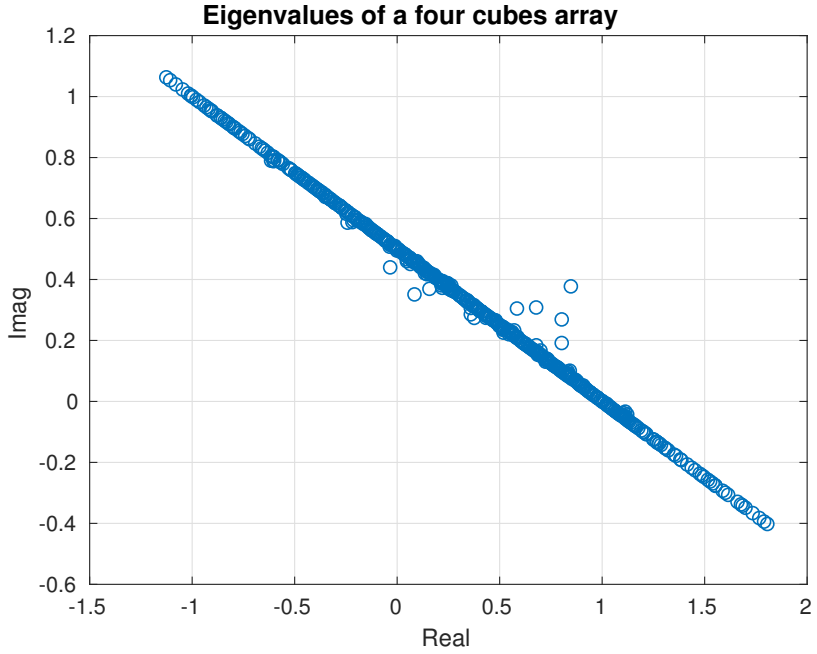


Figure 2: The spectrum of the discretized operator matches relatively well the predictions given by the spectral localization theorem for the continuous operator

Neumann series (see section 3.2.1). In particular, the powerful spectral localization theorems that exist for this kind of operators allow us to build convergent series.

Given a linear system

$$Bu = f \tag{14}$$

consider a fixed point iteration of the form

$$u_{n+1} = H_\mu u_n + \frac{f}{\mu} \tag{15}$$

where the iteration matrix is

$$H_\mu = \frac{\mu I - B}{\mu} \tag{16}$$

Of course, this method converges towards the solution if the spectral radius $\rho(H_\mu) < 1$. It is known that there exists value for μ such that $\rho(H_\mu) < 1$ if the origin of the complex plane is outside the convex envelope of $\sigma(B)$ (spectrum of B). From the eigenvalue localization theorems, we can ascertain

that the spectrum of the matrix of the problem fulfills that condition, so is is possible to find a μ such that the iteration (15) converges. Furthermore, since we have *a priori* information about the spectrum due to the eigenvalue localization problems, we can decide the paramater μ beforehand.

The iteration (15) can be expressed as a Neumann series:

$$u = \sum_{i=0}^{\infty} (H_\mu)^i \frac{f}{\mu} \tag{17}$$

This series is capable of being implemented in a Monte Carlo fashion. Other acceleration methods classically applied to successive approximation methods can be considered.

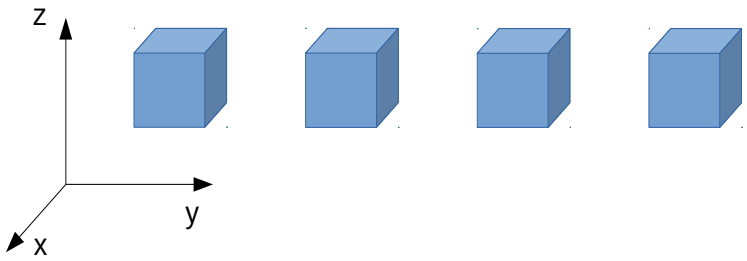
From a physical point view, this methods is interesting as it represents a modified Born series. The Born series, or successive Born approximation, is a method for solving the electromagnetic scattering problem when the scatterer is very small compared with the wavelength. Taking equation (8) and defining the integral operator

$$Lf \equiv k^3 \int_V (\varepsilon(\mathbf{r}) - 1)\overline{\overline{G}}(\mathbf{r}, \mathbf{r}') \cdot f(\mathbf{r}') d^3\mathbf{r}' \tag{18}$$

the successive Born approximation allows to write the solution $\mathbf{E}$ as

$$\mathbf{E} = (I + L + L^2 + \dots)\mathbf{E}_i \tag{19}$$

This is an example of Neumann series on integral operators. In section 3 we will fully explain the meaning of this expression. Here, we should just not that both the born approximation and the fixed point linear iterative method are based on the same principle: repeatedly applying the operator to obtain the solution of the problem.



3 FFT accelerated Ulam-Neumann algorithm

The regular grids involved in our discretization allow the development of efficient algorithms based on the Fast Fourier Transform. The use of FFT to accelerate computations in the framwork of the Method of Moments has been extensively used [15]. In our case, certain discretizations lead to matrices formed by blocks that have a Toeplitz structure. Here, we propose an FFT accelerated version of the Ulam-Neumann algorithm. Our extension of the algorithm consists on considering matrix blocks instead of single element, and employing the fact that these blocks are Toeplitz to accelerate the block computations through FFT. In spite of the large literature on Monte Carlo methods, we have not been able to find the block level generalization of the Ulam-Neumann algorithm in the literature.

In this section, we will first address the topic of matrix multiplications accelerated by FFT, with special attention to the matrices arising from electromagnetic problems. Then, we will describe the blockwise version of the Ulam-Neumann algorithm for matrix inversion. Then, we will combine both things to obtain our FFT accelerated Monte Carlo algorithm.

3.1 FFT accelerated matrix multiplications

3.1.1 The general case

The fundamental principle of these techniques lies in the fact that convolutions can be represented as multiplications in the reciprocal space, under a Fourier transform. It is well known that the convolution of a vector x of length L_x and a vector y of length L_y can be computed as a Hadamard product in the reciprocal space corresponding to a Discrete Fourier Transform of $L_x + L_y - 1$ points.

$$z[n] = x[n] * y[n] \quad \xleftrightarrow{\text{DFT}_{\text{Lx+Ly-1}}} \quad Z[k] = X[k] \circ Y[k] \tag{20}$$

The DFT can be efficiently computed by using the Fast Fourier Trasnform algorithm. The complexity of the DFT is $O(N^2)$, whereas the complexity of the FFT is $O(N \log N)$. This is only possible when the points of the lattice are equispaced. For more about the Fourier transform, see for example [16], [17].

It is possible to represent the convolution of two vectors as the product

of a Toplitz Matrix T and a vector x. The Toeplitz matrix has the structure

$$T = \begin{bmatrix} t_0 & t_{-1} & \dots & t_{-(N-1)} \\ t_1 & t_0 & \dots & t_{-(N-2)} \\ \vdots & \vdots & \ddots & \vdots \\ t_{N-1} & t_{N-2} & \dots & t_0 \end{bmatrix}$$

The product Tx is equivalent to the convolution

$$(Tx)[k] = (t * x)[k] = \sum_{j=0}^{N-1} t_{k-j} x_j \tag{21}$$

By taking a careful look at this matrix, we realize that all the information necessary for computing the convolution is stored in the first column and the first row of the matrix T. In order to accelerate that computation, we should consider vectors whose length is suitable for the FFT algorithm. That could be achieved by padding the vectors with zeros. Let us consider a vector $\hat{t} = (t_0, t_1, ..., t_{N-1}, 0, ..., 0, t_{-(N-1)}, t_{-(N-2)}, ..., t_{-1})^\intercal$. The total length of the vector is n. The integer n should be the minimum power of two ($n = 2^l$) such that $n \geq 2N - 1$. In order to fulfill this condition, $n - (2N - 1)$ zeros are inserted between t_{N-1} and $t_{-(N-1)}$. Let us also consider a vector $\hat{x} = (x_0, x_1, ..., x_{N-1}, 0, ..., 0)$, that is, the x vector with $n - N$ zeros at the end. With this definitions, it is possible to establish that

$$(Tx)[k] = FFT^{-1}(FFT(\hat{t}) \circ FFT(\hat{x}))[k], \quad k = 0, ..., N-1 \tag{22}$$

3.1.2 The electromagnetic scattering case

This kind of acceleration techniques is possible in the framework of Method of Moments due to the fact that the integral equation itself is, in fact, a convolution of the Green's function $\overline{\overline{G}}(\mathbf{r})$ with the function representing the current $\mathbf{J}(\mathbf{r})$ or the electric field $\mathbf{E}(\mathbf{r})$. If the discretization grid is regular, in the sense that the points are equispaced in the three directions of space, the matrix has a Toeplitz structure, and it is possible to accelerate the multiplications through FFT. For example, in the case of a three dimensional system with a completely regular grid the the matrix will have the following structure:

$$Z = \begin{bmatrix} \mathbf{Z}^{xx} & \mathbf{Z}^{xy} & \mathbf{Z}^{xz} \\ \mathbf{Z}^{yx} & \mathbf{Z}^{yy} & \mathbf{Z}^{yz} \\ \mathbf{Z}^{zx} & \mathbf{Z}^{zy} & \mathbf{Z}^{zz} \end{bmatrix} \tag{23}$$

Each one of the nine matrices is a Toeplitz matrix. For example, and without loss of generality, let us focus on the Z_{xx} matrix:

$$\mathbf{Z}^{xx} = \begin{bmatrix} Z_0^{xx} & Z_{-1}^{xx} & \dots & Z_{-(M-1)}^{xx} \\ Z_1^{xx} & Z_0^{xx} & \dots & Z_{-(M-2)}^{xx} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{M-1}^{xx} & Z_{M-2}^{xx} & \dots & Z_0^{xx} \end{bmatrix} \tag{24}$$

Recall that the Toeplitz structure of the matrix, which allows to apply the FFT algorithm, is a result of the regularity of the lattice. In order for the matrix to have this structure, it is necessary that the all the grid is regular. However, in some cases, this requirement may imply the appearance of unnecessary unknowns in the system (there is an unknown associated to each cell). One of the advantages of the discretization of (8) is that it does not require meshing the whole space, but only the points occupied by the dielectric. Meshing the whole space implies considering unknowns associated to empty cells. However, there is a way to overcome this difficulty.

This is depicted in figure (fig.3). The top left image shows the original set of dielectrics that constitute the scatterer. The top right discretization uses the minimum number of cells, since only the volume occupied by the dielectric is discretized. The bottom left image shows a meshing of the whole space. In this case, due to the regularity of the lattice, the system will have a broad Toeplitz structure, since if will be formed by nine big Toeplitz blocks (23). From the point of view of the application of the FFT acceleration this is the most advantageous case, although it implies meshing the whole volume and thus using a large number of ghost cells. The right bottom case is the most interesting from our point of view. Here, the mesh is extended with ghost cells around each one of the scatterers, in such a way that each scatter is embedded in a cuboid mesh (domain) that allows the use of the FFT acceleration. This structure will generate a matrix that is not block Toeplitz, but whose blocks are Toeplitz. Although the extension of the Toeplitz blocks is smaller than in the whole regular meshing, the amount of unknowns is smaller, and we still preserve a part of the Toeplitz advantage. This discretization will be referred locally regular meshing. In general, the matrices arising from a locally regular meshing will have the form

$$Z = \begin{bmatrix} \mathbf{Z}_{0,0} & \dots & \mathbf{Z}_{0,M-1} \\ \vdots & \ddots & \vdots \\ \mathbf{Z}_{M-1,0} & \dots & \mathbf{Z}_{M-1,M-1} \end{bmatrix} \tag{25}$$

where the block $\mathbf{Z}_{ij}$ represents the physical interaction between the local regular grid i and the local regular grid j. The bold type is used to emphasize

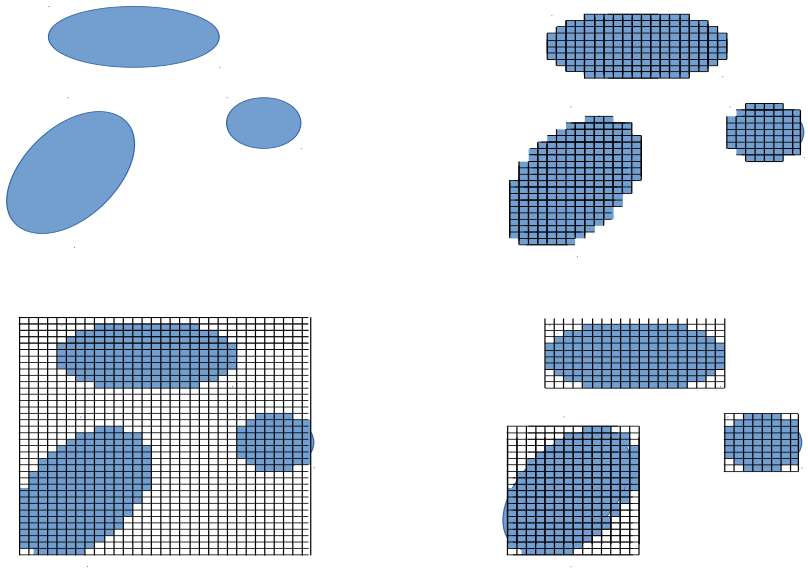


Figure 3: Several discretizations of the same problem. Top left: the original system of three dielectric bodies. Top right: the simplest possible discretization of the problem: only the volume occupied by dielectrics is meshed. Bottom left: a regular meshing on all the volume. Here, ghost cells are added in order for the grid to be regular. Thus, the associated matrix is Toeplitz in a recursive way. Bottom right: the matrix is not block Toeplitz, but the blocks are Toeplitz.

that we are referring to blocks. For example, in the case of fig.3 there are three dielectric bodies, which implies that there are three local regular grids, then $M = 3$.

Blocks $\mathbf{Z}_{ij}$ will have the structure

$$\mathbf{Z}_{ij} = \begin{bmatrix} \mathbf{Z}_{ij}^{xx} & \mathbf{Z}_{ij}^{xy} & \mathbf{Z}_{ij}^{yz} \\ \mathbf{Z}_{ij}^{yx} & \mathbf{Z}_{ij}^{yy} & \mathbf{Z}_{ij}^{yz} \\ \mathbf{Z}_{ij}^{zx} & \mathbf{Z}_{ij}^{zy} & \mathbf{Z}_{ij}^{zz} \end{bmatrix} \tag{26}$$

Remember that, since the unknown at each point is a vector in the 3D space (the electric field) the functions acting on it are of tensorial nature [18], and produce this kind of structures when discretized into a matrix. Physically, the block $\mathbf{Z}_{ij}^{xy}$ represents the effect that the x component of the electric field on the object located in the domain i produces on the y component of the object located on the domain j. Since each one of these subblocks is associated to

a regular grid, their structure will be Toeplitz. Let us consider, without loss of generality, the block $\mathbf{Z}_{ij}^{xy}$:

$$\mathbf{Z}_{ij}^{xy} = \begin{bmatrix} Z_{ij,0}^{xy} & Z_{ij,-1}^{xy} & \cdots & Z_{ij,-(N-1)}^{xy} \\ Z_{ij,1}^{xy} & Z_{ij,0}^{xy} & \cdots & Z_{ij,-(N-2)}^{xy} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{ij,N-1}^{xy} & Z_{ij,N-2}^{xy} & \cdots & Z_{ij,0}^{xy} \end{bmatrix} \tag{27}$$

Due to its Toeplitz structure, the operations involving the block $\mathbf{Z}_{ij}$ will be easily accelerated. Let us consider, for example that our goal is to compute the product of a vector $\mathbf{Z}_{ij}$ and a vector $\mathbf{u}$, where $\mathbf{u}$ is the concatenation of three smaller vectors $\mathbf{u}_x$, $\mathbf{u}_y$ and $\mathbf{u}_z$. These three vectors are of length N. Then,

$$\begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{bmatrix} = \begin{bmatrix} \mathbf{Z}_{ij}^{xx} & \mathbf{Z}_{ij}^{xy} & \mathbf{Z}_{ij}^{yz} \\ \mathbf{Z}_{ij}^{yx} & \mathbf{Z}_{ij}^{yy} & \mathbf{Z}_{ij}^{yz} \\ \mathbf{Z}_{ij}^{zx} & \mathbf{Z}_{ij}^{zy} & \mathbf{Z}_{ij}^{zz} \end{bmatrix} \begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{bmatrix} \tag{28}$$

The classical algorithm for matrix multiplications is applied at the block level, and the product between each one of the subblocks of $\mathbf{Z}_{ij}$ and each one of the subvectors of $\mathbf{u}$ is computed with the aid of the FFT algorithm, taking advantage of the Toeplitz structure of the subblocks. If we define indexes p and q that run over x, y, z, we can write

$$\mathbf{v}_p = \sum_q \mathbf{Z}_{ij}^{pq} \mathbf{u}_q = \sum_q FFT^{-1}(FFT(\hat{z}_{ij}^{pq}) \circ FFT(\hat{u}_q)) \tag{29}$$

where we have defined the vectors

$$\hat{z}_{i,j}^{pq} = (Z_{ij,0}^{pq}, ..., Z_{ij,N-1}^{pq}, 0, ..., 0, Z_{ij,-(N-1)}^{pq}, ..., Z_{ij,-1}^{pq})^\intercal \tag{30}$$

$$\hat{u}_q = (u_{q,0}, ..., u_{q,N-1}, 0, ..., 0)^\intercal \tag{31}$$

with $u_{q,l}$ being the l-th element of the subvector $\mathbf{u}_q$. The zeros had been padded as explained before in such a way that the length of the vectors is a power of 2.

Along this work, when we refer to the blocks of the matrix, we will be referring to the blocks $\mathbf{Z}_{ij}$, not to its subblocks $\mathbf{Z}_{ij}^{pq}$.

3.2 Ulam-Neumann Algorithm

The following algorithm was proposed by Stanislaw Ulam and John von Neumann in the 1940s for solving systems of linear equations. It is based on taking random samples of the Neumann series, a deterministic expression

that evaluates the solution of the system. In order to avoid confusion, it is worth to clarify that the Neumann series is named after Carl Neumann (1832-1925), whereas the Monte Carlo algorithm for stochastically evaluating the Neumann series was developed by John von Neumann (1903-1957) and Ulam in the context of the Manhattan Project.

3.2.1 Mathematical foundations

It is well known that, for $x \in \mathbb{R}$, the following equality holds when $|x| < 1$:

$$\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i \tag{32}$$

This can be obtained, for examples, by taking the Taylor expansion of $1/(1-x)$ around zero. The Neumann series is the equivalent for linear operators. From now on, and for the sake of simplicity, we will focus on matrices defined over the field of complex numbers ($M \in \mathbb{C}^{n \times n}$), which are a particular case of linear operators. We start by defining the eigenvalues and eigenvectors of the matrix. A scalar $\lambda_i \in \mathbb{C}$ and a vector $v_i \in \mathbb{C}^n$ are said to be, respectively, an eigenvalue and an eigenvector of the matrix H is they fulfill

$$Hv_i = \lambda_i v_i \tag{33}$$

The spectrum of a matrix is the set of all its eigenvalues λ_i. We define the spectral radius of a matrix H as the maximum absolute value of its eigenvalues.

$$\rho(H) = \max_i |\lambda_i| \tag{34}$$

The Neumann series of a matrix H is defined as $\sum_{i=0}^{\infty} H^i$. It is possible to prove that, when $\rho(H) < 1$,

$$(I-H)^{-1} = \sum_{i=0}^{\infty} H^i \tag{35}$$

It is, when the spectral radius of the matrix H is smaller than one, then its Neumann series converges towards the inverse of $I - H$. If we consider the problem $x = Hx + b$, the element x_i of the solution vector is computed with the Neumann series as

$$x_i = b_i + \sum_j H_{ij} b_j + \sum_j \sum_k H_{ij} H_{jk} b_k + \sum_j \sum_k \sum_l H_{ij} H_{jk} H_{kl} b_l + \dots \tag{36}$$

This expression will be useful when defining the Monte Carlo estimator of the Neumann series (35).

The Neumann series has a great importance both in pure mathematics and in numerical computing. In pure mathematics, it is applied as a formal tool in linear operator theory and in the study of partial differential equations [19]. In numerical computing, it is the theoretical base of classical methods for solving systems of linear equations, such as Gauss, Jacobi or Richardson methods. The Neumann series can also be understood as a Taylor expansion around zero. In spite of the great insight that this fact provides, it is seldom pointed out in the literature. So, we considered it is interesting to provide here such an explanation.

Let us start with the classical definition of the resolvent of H [19].

$$R_\lambda = (H - \lambda I)^{-1} \tag{37}$$

By considering the change of variables $\lambda = \mu^{-1}$,

$$R_\lambda(H) = (H - \lambda I)^{-1} = -\frac{1}{\lambda}(I - \frac{1}{\lambda}H)^{-1} = -\mu(I - \mu H)^{-1} \tag{38}$$

We define the μ dependent resolvent as $\tilde{R}_\mu = R_{\mu^{-1}} = R_\lambda$. Now, we compute the Taylor expansion of $R_\mu(H)$ around μ_0

$$\tilde{R}_\mu(H) = -\mu \sum_{k=0}^{\infty} \frac{1}{k!} \frac{\partial^k}{\partial \mu^k}(I - \mu H)^{-1}\Big|_{\mu=\mu_0} (\mu - \mu_0)^k \tag{39}$$

After computing the derivative, the factorial $k!$ cancels out and after some manipulations we obtain

$$\tilde{R}_\mu(H) = -\mu \sum_{k=0}^{\infty} (I - \mu_0 H)^{-(k+1)} H^k (\mu - \mu_0)^k \tag{40}$$

If we set $\mu_0 = 0$, we obtain

$$\tilde{R}_\mu(H) = -\sum_{k=0}^{\infty} \mu^{k+1} H^k \tag{41}$$

And, after setting $\mu = 1$, we will recover the the Neumann series.

Now, we will prove that the Neumann series converges when the condition $\rho(H) < 1$ is fulfilled. This can be proven from the convergence conditions for the Taylor expansion. It is well known [20] that the Taylor series (39) is convergent only in a disk centered at μ_0 with radius of convergence r. This is known as disk of convergence. We say that a function is analytic within

the disk of convergence, meaning that it can be given as a power series. According to complex analysis, a function is analytic on a disk when it is holomorphic, it is, when it is differentiable at every point on the disk. In our case, the function R_μ is differentiable everywhere except at its singular points, which are the points where the inverse of $(I - \mu H)$ does not exist. Obviously, it happens for the values μ_i that make $I - \mu H$ singular, i.e. for some v_i,

$$(I - \mu_i H)v_i = 0 \tag{42}$$

and clearly, this values μ_i correspond to the eigenvalues of H as described on (33).

The conclusion is that the Taylor expansion (39) for $R_\mu(H)$ converges on a disk centered at μ_0 if the disk does not contain any of the points $\mu_i = 1/\lambda_i$, where λ_i are the eigenvalues of H. So we can write the disk of convergence as

$$|\mu - \mu_0| < |\frac{1}{\lambda_i} - \mu_0|, \quad \forall i \tag{43}$$

In particular, for the case were we are expanding around $\mu_0 = 0$ we can write

$$|\mu| < \frac{1}{\max_i |\lambda_i|} \tag{44}$$

and if we set $\mu = 1$ to get the Neumann series, the convergence condition can be written as

$$\max_i |\lambda_i| = \rho(H) < 1 \tag{45}$$

So, we have shown that the Neumann series 35 converges only if the spectral radius of the matrix H is less than one.

Let us try to give a general vision about the mathematical tools we have seen until now. The Neumann series belong to the realm of operator theory. In a non rigorous mathematical way, we can consider that it is the Taylor expansion of the inverse of a certain operator centered at zero. We have deduced it in this way. When the Neumann series is applied to the scattering operator to solve the scattering problem, it is known as Born series or method of successive Born approximations.

Besides, we can consider that the Neumann series is an expansion on the Krylov space [21]. The Krylov space of a matrix A is the space expanded by I, A, A^2, A^3, etc. Modern iterative methods, such as GMRES, find approximations of the solution of a SLAE as vectors in the Krylov space ($x \approx \sum \alpha_i A^i$). This method finds the coefficients α_i in such a way that a few terms are enough to approximate the solution very well. Usually,

the coefficients are computed iteratively. The Neumann series, in which the Ulam-Neumann Monte Carlo method is based, can be considered an expansion in the Krylov space where all the coefficients are 1. Although this series converges slowly, it has the advantage that all the coefficients are known beforehand. This allows to implement the Ulam-Neumann algorithm, since this algorithm takes samples of an expression (the Neumann series) whose form should be fully known from the very beginning (36).

3.2.2 Description of the Ulam-Neumann algorithm

Now, we will introduce the Ulam-Neumann algorithm, based on randomly sampling (35). For more details about this algorithm, see for example [4]. Let us consider a linear system of equations of the form:

$$Ax = b, \qquad A \in \mathbb{C}^{n \times n} \tag{46}$$

By defining $H = I - A$, we can write the system as $(I - H)x = b$. The inverse $A^{-1} = (I - H)^{-1}$ can be found by means of the Neumann series if H fulfills the convergence condition. In the context of iterative solvers [21] the matrix H is known as the iteration matrix.

The Monte Carlo algorithm computes an approximation of the functional $J = \langle h, c \rangle$. The approximation is denoted as $\theta[h]$. We can reasonable estimate J by compute N instances of $\theta[h]$ and averaging them [4]:

$$J \approx \frac{1}{N} \sum_{s=1}^{N} \theta_s[h] \tag{47}$$

In order to build the estimator $\theta[h]$, we start by defining a Markov chain of length $k + 1$.

$$\gamma_k = \xi_0 \to \xi_1 \to ... \to \xi_k \tag{48}$$

where $\xi_i = 0, ..., n - 1$. The probability of the first element of the chain to be a certain state i is noted as P_i, and the probability of transitioning from a state i to a state j is noted as $P_{i,j}$. Also, the termination probability T_i represent the probability of the chain to terminate after the state i

The estimator $\theta[h]$ is defined as

$$\theta[h] = \frac{h_{\xi_0}}{P_{\xi_0}} \sum_{i=0}^{k} W_i b_{\xi_i} \tag{49}$$

$$W_0 = 1 \tag{50}$$

$$W_i = W_{i-1} \frac{H_{\xi_{i-1},\xi_i}}{P_{\xi_{i-1},\xi_i}} \tag{51}$$

Note that we are considering that the entries of matrices and vectors go from 0 to $n-1$. The transition probabilities are usually encoded on a matrix P, where the element $P_{i,j}$ is the transition probability from i to j. The matrix P should obey the condition

$$\sum_{j=0}^{n-1} P_{i,j} < 1, \qquad \forall i = 0 \dots n-1 \tag{52}$$

The termination probability T_i is precisely

$$T_i = \sum_{j=0}^{n-1} P_{i,j} \tag{53}$$

Let us now take a close look on the similarity between the Neumann series expansion for one element of the solution (36) en the estimator (49). The estimator can be understood as a sample of equation (36), where we randomly select a set of the summation indices, and where matrix elements have been weighted according to their probability to be chosen. In fact, if we expand the expression for the estimator, the ressemblance becomes obvious:

$$\theta[h] = \frac{h_{\xi_0}}{P_{\xi_0}}(b_{\xi_0} + \frac{H_{\xi_0,\xi_1}}{P_{\xi_0,\xi_1}} b_{\xi_1} + \frac{H_{\xi_0,\xi_1} H_{\xi_1,\xi_2}}{P_{\xi_0,\xi_1} P_{\xi_1,\xi_2}} b_{\xi_2}) + \dots \tag{54}$$

It is possible to prove that the expected value of this estimator is $J = \langle h, c \rangle$ and that its variance is bounded [5]. However, this issues will not be addressed in this section. In the following section, a block level version of this algorithm is proposed, and the proof of convergence will be addressed there. Here we just need to take into consideration one of the main points of the article [5]: the estimator (49) converges to the solution if $\rho(H^*) < 1$, where H^* is a matrix defined as

$$H^*_{i,j} = \frac{H^2_{i,j}}{P_{i,j}} \tag{55}$$

This implies that the convergence does not only depend on the properties of H, but also on the properties of the transition matrix. A transition matrix that ensures convergence is not always trivial to find. Nevertheless, [5] explains how a P matrix fulfilling $\rho(H^*) < 1$ can be easily computed providing that $\|H\|_\infty < 1$. Unfortunately, not all the matrices H fulfill this condition. In some cases, however, the system can be preconditioned in such a way that the iteration matrix H fulfills $\|H\| < 1$. Once this condition is fulfilled, according to [5], it will be trivial to compute a transition matrix P that ensures convergence of the algorithm.

Consider again a system of the form (46). We say that the system is preconditioned when we modify it by applying a preconditioning matrix E:

$$EAx = Eb \tag{56}$$

We expect this transformation to simplify the solution of the linear system. Now, the iteration matrix is $H = I - EA$. Usually, preconditioning is addressed to reduce the condition number of the system matrix [21]. In this particular case, we are interested into a preconditioning that ensures $||H|| < 1$, in such a way that the Monte Carlo algorithm easily converges. A very similar or equivalent preconditioning method was proposed by Alexandrov and Dimov [22].

Let A be a strictly diagonally dominant matrix. We say that a matrix is strictly diagonally dominant when

$$|A_{ii}| > \sum_{j \neq i} |A_{ij}|, \qquad \forall i = 0 \dots n-1 \tag{57}$$

We define a matrix E in the following way

$$E_{ij} = \begin{cases} \frac{1}{A_{ij}} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{58}$$

This matrix is applied as a preconditioner to our system, thus obtaining EA as the system matrix and $H = I - EA$ as the iteration matrix. Computing the norm of $H = I - EA$ gives

$$||H|| = \max_i \Big(\sum_{j \neq i} \left| \frac{A_{ij}}{A_{ii}} \right| + \left| 1 - \frac{A_{ii}}{A_{ii}} \right| \Big) = \max_i \Big(\sum_{j \neq i} \left| \frac{A_{ij}}{A_{ii}} \right| \Big) < 1 \tag{59}$$

were we have used that A is strictly diagonally dominant.

In order for this preconditioning to work as described, we are requiring the system matrix A to be diagonally dominant. Certainly, this is a very strong condition. As we will see later, the system matrices associated with the physical problem we are addressing have an structure that approaches the diagonal dominance requirement since, the absolute value of the element tends to decrease as we move away from the diagonal. Still, diagonal dominance is a very strict requirement. Nevertheless, in the following section we will introduce the blockwise version of this algorithm. Thus, the associated preconditioner will require the matrix to be diagonally dominant in a block level way. This condition is easier to fulfill, and it better matches with the physical nature of the problem. We will see a numerical example of this in section 4.

3.3 Blockwise Ulam-Neumann Algorithm

In this section, we propose an extension of the Ulam-Neumann algorithm to the block level case. As fas as we know, although there are works addressing the block level structure of the matrix [23], they do not implement the Ulam-Neumann algorithm in a block level way. The idea here is fairly simple: we solve a system of linear equations by sampling the sum (35) by the Ulam-Neumann algorithm, but considering matrix blocks instead of matrix elements. In this section we describe this algorithm and we extended the proofs of convergence developed by [5] to the blockwise case. This algorithm shares its mathematical foundations with the plain version (element wise). So, for that matter, we refer to section 3.2.1.

Applying a blockwise version of the algorithm can have several benefits. From a parallel computing point of view, we are reducing the granularity of our algorithm. This has two main potential advantages:

- Efficiently obtaining the elements of the matrix. Whether the matrix blocks are stored in memory or they are recomputed only when necessary, the block level implementation of the algorithm can reduce the access time to the blocks. If the element are accessed in memory, we can benefit from spatial locality. Optimally, the data structure representing the matrix should be such that the elements of a single block are contiguously stored in memory. If the matrix elements are computed only when needed, we will benefit from the fact that the algorithm that computes matrix elements is more efficient for computing a relatively large amount of this elements rather than a s single one.

- Efficient matrix-vector product. The matrix operations can be efficiently performed by employing good numerical libraries, fast exponentiation or, in our case, the FFT accelerated product, possible due to the Toeplitz structure of the blocks.

3.3.1 Description of the blockwise Ulam-Neumann algorithm

We start again considering an expansion of $(I-H)^{-1}$ in terms of the Neumann series (35). Let us suppose that the linear system is written as $u = Hu + f$. The matrix H is divided into blocks of size $N \times N$. These blocks are noted as $\mathbf{H}_{i,j}$, with $i, j = 0, ..., M-1$ (so $MN \times MN$ is the size of the matrix H). The vector f is also divided into M subvectors of length N: $f = (\mathbf{f}_0, \mathbf{f}_1, ..., \mathbf{f}_{M-1})^\intercal$. In a similar way, the unknown vector u can be written as $u = (\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{M-1})^\intercal$.

On the previous section, we wrote the expression for a single element of the solution vector as an expansion based on the Neumann series (36). In a similar, for subvector $\mathbf{u}_i$ of the solution we can write

$$\mathbf{u}_i = \mathbf{f}_i + \sum_j \mathbf{H}_{ij}\mathbf{f}_j + \sum_j \sum_k \mathbf{H}_{ij}\mathbf{H}_{jk}\mathbf{f}_k + \sum_j \sum_k \sum_l \mathbf{H}_{ij}\mathbf{H}_{jk}\mathbf{H}_{kl}\mathbf{f}_l + \dots \quad (60)$$

As in the classical case, we will take random samples of this series in order to get an estimation of $\mathbf{u}_i$. Regarding the size of the blocks, we have said the taking blocks of a certain size instead of single elements can make our algorithm more computationally efficient. However, we should not lose sight of the fact that, if the blocks are too big, the Monte Carlo method will lose its purpose. The idea of any stochastic approach is randomly selecting a few elements a the problem. Thus, the size of the blocks should be very small compared with the size of the matrix ($N << M \times N$), in such a way that by sampling (60) we are still considering a small part of our system.

Following this train of thought, since the blocks should be 'big' in absolute terms for the blockwise Ulam-Neumann algorithm to be efficient, and the matrix should be big compared with the size of the blocks for the Monte Carlo approach to be worth, the conclusion is that this algorithm is only suitable for huge systems of equations.

Let us come back to the algorithm. Our goal is again to compute an estimator for a functional $J = \Omega^\intercal u$, where Ω is an arbitrary matrix of size MNxN. Since $N << MN$ for the aforementioned reasons, Ω is going to be a 'tiny' but tall matrix. The matrix Ω is also divided in blocks of size NxN, that are noted as $\mathbf{\Omega}_i$. For clarity, we depict the Ω matrix in the plain and block notations:

$$\Omega = \begin{bmatrix} \Omega_{0,0} & \dots & \Omega_{0,N-1} \\ \vdots & & \vdots \\ \Omega_{MN-1,0} & \dots & \Omega_{MN-1,N-1} \end{bmatrix} = \begin{bmatrix} \mathbf{\Omega}_0 \\ \vdots \\ \mathbf{\Omega}_{M-1} \end{bmatrix} \quad (61)$$

Then, $J = \Omega^\intercal u$ will be a vector of length N. In particular if $\mathbf{\Omega}_i = I_N$ (the identity matrix of size N) and $\mathbf{0}$ otherwise, then $J = \mathbf{u}_i$.

The functional J can be computed by an estimator consisting on statistically sampling the Neumann series. First of all, let us consider the Markov chain of length $k + 1$.

$$\gamma_k = \xi_0 \to \xi_1 \to ... \to \xi_k \quad (62)$$

where $\xi_i = 0, ..., M - 1$, it is, the random variable takes an integer value between 0 and $M - 1$. The probability for the first element of the chain to

take a certain value a is noted as P_a. The transition probability of obtaining a state c when the current state is b is noted as $P_{b,c}$.

Then, the estimator of the functional J for a certain Ω takes the form

$$\Theta(\Omega) = \frac{\mathbf{\Omega}^{\intercal}_{\xi_0}}{P_{\xi_0}} \sum_{j=0}^{k} \mathbf{W}_j \mathbf{f}_{\xi_j} \tag{63}$$

$$\mathbf{W}_0 = I_N \tag{64}$$

$$\mathbf{W}_j = \mathbf{W}_{j-1} \frac{\mathbf{H}_{\xi_{j-1},\xi_j}}{P_{\xi_{j-1},\xi_j}} \tag{65}$$

It is necessary to prove that this estimator actually converges towards the solution. The proof consists of two parts. First, we should prove that the mean value of this estimator is the solution. Second, we should prove that the variance is bounded. In order to do so, we generalize a part of the work [5] for the block level case.

For the sake of simplicity, we will limit ourselves, for cases where the transition matrix P is easy to find. In order to do that, we will require the iteration matrix H to fulfill some supplementary conditions. These supplementary conditions are:

1. The blocks $\mathbf{H}_{ij}$ are Toeplitz. This is certainly a very restrictive condition, however, it is intrinsically a characteristic of the physical case we are considering, and is what allows us to speed up matrix multiplications thanks to FFT. We will use this fact to simplify the proof of convergence.

2. The blocks satisfy the condition

$$\sum_{j=0}^{M-1} \|\mathbf{H}_{i,j}\| < 1 \quad \forall i \tag{66}$$

Note that the second condition is more strict than simply $\|H\| < 1$. It is possible to force the iteration matrix to fulfill that condition by properly adjusting the constants involved in the algorithm [22].

First of all, we prove that the mean of the estimator is the value $J = \Omega^{\intercal} u$:

$$
\begin{aligned}
E\Big\{\Theta(\Omega)\Big\} &= E\Big\{\frac{\mathbf{\Omega}_{\xi_0}^{\intercal}}{P_{\xi_0}}\Big(I_N\mathbf{f}_{\xi_0}+\frac{\mathbf{H}_{\xi_0,\xi_1}}{P_{\xi_0,\xi_1}}\mathbf{f}_{\xi_1}+\frac{\mathbf{H}_{\xi_0,\xi_1}\mathbf{H}_{\xi_1,\xi_2}}{P_{\xi_0,\xi_1}P_{\xi_1,\xi_2}}\mathbf{f}_{\xi_2}+\dots\Big)\Big\}\\
&= \sum_{i=0}^{M-1}P_i\frac{\mathbf{\Omega}_i^{\intercal}}{P_i}\mathbf{f}_i+\sum_{i,j=0}^{M-1}P_iP_{i,j}\mathbf{\Omega}_i^{\intercal}\frac{\mathbf{H}_{i,j}}{P_iP_{i,j}}\mathbf{f}_j+\sum_{i,j,k=0}^{M-1}P_iP_{i,j}P_{j,k}\mathbf{\Omega}_i^{\intercal}\frac{\mathbf{H}_{i,j}\mathbf{H}_{j,k}}{P_iP_{i,j}P_{j,k}}\mathbf{f}_k+\dots\\
&= \Omega^{\intercal}\sum_{i=0}^{\infty}H^if = \Omega^{\intercal}u
\end{aligned}
\tag{67}
$$

Even if the mean of the estimator is J, it is necessary to prove that it converges, it is, that the variance is bounded. We reproduce the Lemma 3.3 and Theorem 3.4 from [5] for the block case. For more details, see the aforementioned work.

Thanks to condition 2, it is possible to find a transition matrix such that

$$
\sum_{i=0}^{M-1}\frac{||\mathbf{H}_{i,j}||^2}{P_{i,j}}<1 \tag{68}
$$

A way to build this P matrix would be simply taking $P_{ij}=\|\mathbf{H}_{ij}\|$, which is the block level equivalent to Monte Carlo Almost Optimal [4]. Now, we will prove that the P matrix fulfilling (68) ensures that the variance of the blockwise Ulam-Neumann Monte Carlo algorithm is bounded.

Since $\Omega^{\intercal}u$ is a vector, we should use here the covariance matrix, that is the vector equivalent of the scalar variance. Let us consider that γ_k is the Markov chain of length $k+1$. Then, we compute

$$
\begin{aligned}
Var(\mathbf{W}_k\mathbf{f}_{\xi_k}) &= E\big\{(\mathbf{W}_k\mathbf{f}_{\xi_k}-E(\mathbf{W}_k\mathbf{f}_{\xi_k}))(\mathbf{W}_k\mathbf{f}_{\xi_k}-E(\mathbf{W}_k\mathbf{f}_{\xi_k})^{\intercal}\big\}\\
&= E\big\{(\mathbf{W}_k\mathbf{f}_k)(\mathbf{W}_k\mathbf{f}_k)^{\intercal}\big\}-E\big\{(\mathbf{W}_k\mathbf{f}_k)\big\}E\big\{(\mathbf{W}_k\mathbf{f}_k)^{\intercal}\big\}
\end{aligned}
\tag{69}
$$

We focus on the first term of the sum, since it is easy to prove that the second is bounded. We denote $f_{max}=\max_i(|\mathbf{f}_i|)$ and $c_{max}=\max_i(\sum_{j=0}^{(M-1)}||\mathbf{H}_{i,j}||^2/P_{i,j})$. Thanks to condition 2, we know that $c_{max}<1$. Then, the norm of the first term of the covariance matrix is

$$
\begin{aligned}
&\|E\big\{(\mathbf{W}_k\mathbf{f}_k)(\mathbf{W}_k\mathbf{f}_k)^\intercal\big\}\| \\
&= \|\sum_{\xi_0=0}^{M-1}\cdots\sum_{\xi_k=0}^{M-1} P_{\xi_0,\xi_1}P_{\xi_1,\xi_2}\dots P_{\xi_{k-1},\xi_k}\frac{(\mathbf{H}_{\xi_0,\xi_1}\mathbf{H}_{\xi_1,\xi_2}\dots\mathbf{H}_{\xi_{k-1},\xi_k}\mathbf{f}_{\xi_k})(\mathbf{H}_{\xi_0,\xi_1}\mathbf{H}_{\xi_1,\xi_2}\dots\mathbf{H}_{\xi_{k-1},\xi_k}\mathbf{f}_{\xi_k})^\intercal}{(P_{\xi_0,\xi_1}P_{\xi_1,\xi_2}\dots P_{\xi_{k-1},\xi_k})^2}\| \\
&\leq \sum_{\xi_0=0}^{M-1}\sum_{\xi_1=0}^{M-1}\cdots\sum_{\xi_k=0}^{M-1} P_{\xi_0,\xi_1}P_{\xi_1,\xi_2}\dots P_{\xi_{k-1},\xi_k}\frac{\|\mathbf{H}_{\xi_0,\xi_1}\|\dots\|\mathbf{H}_{\xi_{k-1},\xi_k}\|\|\mathbf{f}_{\xi_k}\|\|\mathbf{f}_{\xi_k}^\intercal\|\|\mathbf{H}_{\xi_{k-1},\xi_k}^\intercal\|\dots\|\mathbf{H}_{\xi_0,\xi_1}^\intercal\|}{(P_{\xi_0,\xi_1}P_{\xi_1,\xi_2}\dots P_{\xi_{k-1},\xi_k})^2} \\
&= \|\mathbf{f}_{\xi_k}\|^2\sum_{\xi_0=0}^{M-1}\frac{\|\mathbf{H}_{\xi_0,\xi_1}\|^2}{P_{\xi_0,\xi_1}}\cdots\sum_{\xi_k=0}^{M-1}\frac{\|\mathbf{H}_{\xi_{k-1},\xi_k}\|^2}{P_{\xi_{k-1},\xi_k}} \leq f_{max}c_{max}^k
\end{aligned}
\tag{70}
$$

Here, we have used that the infinite norm is submultiplicative and that $||A|| = ||A^\intercal||$ if A is Toeplitz. Since $c_{max} < 1$, $f_{max}c_{max}^k$ tends to zero as k tends to infinity. From here, it is possible to prove that the variance of the estimator $\Theta(\Omega)$ goes to zero as k tends to infinity, since the elements of the sum tend to zero in an exponential way. With this, we have proven that blockwise Ulam-Neumann algorithm converges towards the solution under the specified circumstances.

3.3.2 Preconditioning

We have seen before that linear system of equations whose system matrix is diagonally dominant can be solved by the Ulam-Neumann algorithm after applying a certain preconditioning. Here, we will define an equivalent preconditioning for the blockwise case. First of all, we need to introduce some definitions. Considering the block level partition that we have used before, we say that a matrix A is *strictly block diagonally dominant* if

$$
\sum_{j\neq i}\|\mathbf{A}_{ij}\| < \|\mathbf{A}_{ii}\|, \qquad \forall i = 0\dots N-1 \tag{71}
$$

Note that this is a less strict condition for the matrix than being diagonally dominant in the element level case. So, a matrix can be strictly block diagonally dominant and not being strictly diagonally dominant. We will define the dominance factor d_i as

$$
d_i = \frac{\|\mathbf{A}_{ii}\|}{\sum_{j\neq i}\|\mathbf{A}_{ij}\|} \tag{72}
$$

Obviously, when the matrix is strictly block diagonally dominant, $d_i > 1$.

Let us now define the block level norm of a matrix as

$$\|A\|_{\infty}^{\text{Block}} = \max_i \Big(\sum_j \|\mathbf{A}_{ij}\|_{\infty} \Big) \tag{73}$$

Here, we have based the block level norm in the infinite norm of a matrix. However, we could have used any norm. As usually, if we do not explicitly state which norm we are using, we assume it is the infinite norm: $||A||^{\text{Block}} = ||A||_{\infty}^{\text{Block}}$.

Implicitly, we have already use that norm in equation (66), a necessary requirement for proving that the variance is bounded and the estimator converges. So, according to (66), we need that $||H||^{\text{Block}} < 1$ in order to be sure that the algorithm works. Note that the relation between $\|H\|^{\text{Block}} < 1$ and convergence is not biconditional. It is, the matrix could not fulfill this condition and the algorithm may still converge. However, when the condition is satisfied, not only we are sure that it will converge under a certain transition matrix, but also we know a trivial way to compute such matrix.

Back to our problem, let us assume that we have a system $Au = b$ where A is strictly block diagonally dominant. As we will see later, this is a reasonable condition for our physical problems. For the blockwise Ulam-Neumann algorithm to converge, we would need a preconditioning $EAu = Eb$ such that

$$\|H\|^{\text{Block}} = \|I - EA\|^{\text{Block}} < 1 \tag{74}$$

The matrix E formed by $N \times N$ blocks of size $M \times M$. Following our notation, the blocks are noted as $\mathbf{E}_{ij}$. The matrix has the structure

$$\mathbf{E}_{ij} = \begin{cases} \alpha_i \mathbf{A}_{ij}^{-1} & \text{if } i = j \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{75}$$

where $\mathbf{0}$ is the matrix of zeros of size $M \times M$ and α is a certain parameter. Since the blocks of A are very small compared with the total size of the matrix, computing the inverse of $\mathbf{A}_{ii}$ is not an excessive computational burden.

Now we should prove that this preconditioning, for a certain set of values for α_i, actually ensures that the convergence condition is met. So, computing the block level norm of H

$$\|H\|^{\text{Block}} = \|I - EA\|^{\text{Block}} = \max_i \Big(\sum_{j \neq i} \|\alpha_i \mathbf{A}_{ii}^{-1} \mathbf{A}_{ij}\| + \|\mathbf{I}_M - \alpha_i \mathbf{A}_{ii}^{-1} \mathbf{A}_{ii}\| \Big) \tag{76}$$

where $\mathbf{I}_M$ is the identity matrix of size M. By using the submultiplicative property of the norm ($\|AB\| \leq \|A\|\|B\|$) we obtain

$$\|H\|^{\text{Block}} \leq \max_i \left(|\alpha_i| \|\mathbf{A}_{ii}\| \sum_{j \neq i} \|\mathbf{A}_{ij}\| + |1 - \alpha_i| \right) \tag{77}$$

At this point, we can make several choices for the paramter α_i. In particular, we propose

$$\alpha_i = \frac{1}{\|\mathbf{A}_{ii}\| \|\mathbf{A}_{ii}^{-1}\|} \tag{78}$$

Due to the submultiplicative property of the matrix norm, $\alpha_i \leq 1$. Instering this into the previous equation gives:

$$\|H\|^{\text{Block}} \leq \max_i \left(\frac{1}{d_i} + 1 - \frac{1}{\|\mathbf{A}_{ii}\| \|\mathbf{A}_{ii}^{-1}\|} \right) \tag{79}$$

The convergence condition is $\|H\|^{\text{Block}} < 1$. This can be written as

$$\left(\frac{1}{d_i} + 1 - \frac{1}{\|\mathbf{A}_{ii}\| \|\mathbf{A}_{ii}^{-1}\|} \right) < 1, \qquad \forall i \tag{80}$$

which is equivalent to

$$d_i > \|\mathbf{A}_{ii}\| \|\mathbf{A}_{ii}^{-1}\|, \qquad \forall i \tag{81}$$

The convergence condition (81) is not always going to be true. However, as we will see later, it is likely to be be fulfilled in the physical problems we are addressing. At this moment, though, we can glimpse some aspects about the condition (81), while summarizing what we have done until this moment:

The condition (81) tells us that, if the dominance factor is larger than $\|\mathbf{A}_{ii}\| \|\mathbf{A}_{ii}^{-1}\|$ for all the block level rows i of the matrix, then the block level norm of the iteration matrix H will be less than one. According to the reasoning presented in section 3.3.1, that consist on the generalization of the work [5], this means that the blockwise Ulam-Neumann algorithm will converge towards the solution under the effect of some well known transition matrices (for example, Monte Carlo Almost Optimal).

The condition (81) suggests that, if the dominance factor i is very large, our algorithm will easily converge. This is true. However, even if the algorithm converges, it may be prohibitively slow. Let us imagine, for example, that d_i and $\|\mathbf{A}_{ii}\| \|\mathbf{A}_{ii}^{-1}\|$ are very large numbers, let us say $2 \cdot 10^6$ and 10^6, respectively. In this case, the convergence condition will be satisfied, but $\|H\|^{\text{Block}} = 0.9999995$. This means that the algorithm will converge because

the block level norm of the iteration matrix is less than one, but, since it is very close to one, we expect the convergence to be very slow.

Then, ideally, we would like the block level of H to not to be close to one, and this is why the factor $\|\mathbf{A}_{ii}\|\|\mathbf{A}_{ii}^{-1}\|$ is important. Even if the convergence condition is met, we would like this factor to be as small as possible, in order to reduce the block level norm of H and accelerate convergence. However, we known that it is going to be always larger than one (using the submultiplicative property of the norm, $\|\mathbf{A}_{ii}\|\|\mathbf{A}_{ii}^{-1}\| \geq \|\mathbf{A}_{ii}\mathbf{A}_{ii}^{-1}\| = 1$). It seems that we can only hope that this factor is not much larger than one.

3.3.3 Toeplitz structure preserving preconditioning

One of the disadvantages of the previous preconditioning strategy is that it does not preserve the Toeplitz structure of matrix blocks, thus preventing us to accelerate matrix multiplications through FFT. The following preconditioning strategy does preserve this structure, but it is harder to define clear convergence conditions. However, it is also easier to apply. The key idea is to multiply every block by single scalar instead of another block.

In this case, the block $\mathbf{E}_{ij}$ of the matrix is

$$\mathbf{E}_{ij} = \begin{cases} \beta_i \frac{1}{\|\mathbf{A}_{ij}\|}\mathbf{I}_M & \text{if } i = j \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{82}$$

After applying this, and using the definition of the dominance factor d_i, the block level norm of the iteration matrix $H = I - EA$ will be

$$\|H\|^{\text{Block}} = \max_i \left(\frac{|\beta_i|}{d_i} + \|\mathbf{I}_M - \beta_i \frac{\mathbf{A}_{ii}}{\|\mathbf{A}_{ii}\|}\| \right) \tag{83}$$

In some cases, it will be possible to adjust the constant β_i to fulfill the condition $\|H\|^{\text{Block}} < 1$. We will see some examples later, but in general, it will be desirable that the block level dominance of matrix A is as large as possible ($d_i >> 1$).

3.3.4 FFT accelerated Implementation

The blockwise Ulam-Neumann algorithm achieves its efficiency through performing the block matrix vector products in an efficient way. In our case, the Toeplitz structure of the blocks, that results from the physical discretization of the problem, allows to accelerate the matrix-vector products through the Fast Fourier transform algorithm. Let us consider that the the block $\mathbf{H}_{ij}$ of

the matrix has the form

$$\mathbf{H}_{ij} = \begin{bmatrix} H_{ij,0} & H_{ij,-1} & \dots & H_{ij,-(N-1)} \\ H_{ij,1} & H_{ij,0} & \dots & H_{ij,-(N-2)} \\ \vdots & \vdots & \ddots & \vdots \\ H_{ij,N-1} & H_{ij,N-2} & \dots & H_{ij,0} \end{bmatrix} \tag{84}$$

As we said before, we only modify the Monte Carlo computation of the inverse of the diagonally dominant matrix $\hat{B}$ by using an FFT accelerated blockwise method instead of a classical Monte Carlo method. Here, we descibe the algorithm for solving the linear system $u = Hu+f$, as the extension for finding $H^{-}1$ is easy. Let us again consider a Markov chain γ_k of length $k+1$ associated with a transition matrix P. Similarly to the previous section, the estimator of the product $J = \Omega^{\intercal}u$ is

$$\Theta(\Omega) \quad = \frac{\mathbf{\Omega}^{\intercal}_{\xi_0}}{P_{\xi_0}} \sum_{j=0}^{k} \mathbf{w}_j \tag{85}$$

Where the vector $\mathbf{w}_j$ is recursively defined with the aid of the vectors $\mathbf{w}_0^j$, $\mathbf{w}_1^j$, $\dots$, $\mathbf{w}_{j-1}^j$:

$$\begin{aligned} \mathbf{w}_0^j &= \mathbf{f}_{\xi_j} \\ &\vdots \\ \mathbf{w}_i^j &= \frac{\mathbf{H}_{\xi_{j-i},\xi_{j-i+1}}}{P_{\xi_{j-i},\xi_{j-i+1}}} \mathbf{w}_{i-1}^j \\ &\vdots \\ \mathbf{w}_j &= \frac{\mathbf{H}_{\xi_0,\xi_1}}{P_{\xi_0,\xi_1}} \mathbf{w}_{j-1}^j \end{aligned} \tag{86}$$

The element $\mathbf{w}_i^j$ is efficiently computed as

$$\mathbf{w}_i^j = \frac{1}{P_{\xi_{j-i},\xi_{j-i+1}}} (FFT^{-1}(FFT(\hat{h}_{\xi_{j-i},\xi_{j-i+1}}) \circ FFT(\hat{w}_{i-1}^j)))_N \tag{87}$$

Where we are using the auxiliary vectors

$$\hat{h}_{i,j} = (H_{ij,0}, \dots, H_{ij,N-1}, 0, \dots, 0, H_{ij,-(N-1)}, \dots, H_{ij,-1})^{\intercal} \tag{88}$$

and $\hat{w}_i^j$, which corresponds to $\mathbf{w}_i^j$ with the appropriate number of zeros appended at the end. Note that we have also padded zeros within the vector $\hat{h}_{i,j}$ according to the principles previously described.

It should also be noted that the Toeplitz structure of the blocks is not only useful for accelerating the multiplications through FFT, but also allows to store the matrix efficiently, as any block is determined just by the first row and the first column.

The case arising from our physical problem is slightly more complex than this case. In the example we have previously seen, we are considering that the blocks $\mathbf{H}_{ij}$ are Toeplitz. In the real case, the blocks $\mathbf{Z}_{ij}$ are formed by 9 subblocks that represent the nine components of the tensor function (26). One may consider the possibility of setting the block level at the block components of $\mathbf{Z}_{ij}$, it is, the blocks $\mathbf{Z}_{ij}^{xx}$, $\mathbf{Z}_{ij}^{xy}$ and so on. In that way, the blocks considered by the Ulam-Neumann algorithm would be genuinely Toeplitz. The problem with this approach comes when we study the block level diagonal dominance of the matrix. Considering $\mathbf{Z}_{ij}$ as the constitutive blocks of our matrix results in block level diagonal dominance, which allows us to implement the preconditioning strategies described before. We will come back to this topic in section 4.

Next, we will give an algorithmic description of the blockwise Ulam-Neumann method, focused on the case matrices arising from the Method of Moment formulation of the electromagnetic scattering problem.

ALGORITHM: Preconditioning

Input: Z, e_i

Output: e_s (scattered field)

Compute E ▷ Preconditioning matrix

$\hat{Z} = EZ$ ▷ Preconditioned Z

$\hat{e}_i = Ee_i$

$e_s = \text{BlockMC}(\hat{Z},\hat{e}_i)$

And the BlockMC(,) algorithm for solving the system $Ax = b$, which corresponds to the application of the blockwise Ulam-Neumann method, is:

ALGORITHM: BlockMC(,)

Input: A, f, Monte Carlo execution parameters

Output: u

$H = I - A$

Initialize u as the solution vector (all zeros).

Initialize *count* as the counting vector ().

s=1

while $s < N$ **do**

Generate Markov chain γ_k

$counting[\xi_0] = counting[\xi_0] + 1$
while $j < k$ **do**
$\mathbf{w}_0 = \mathbf{f}_{\xi_j}$
Compute $\mathbf{w}_j$ using recurrence (eq.86) (with FFT)
$j = j + 1$
end while
Compute $\Theta(\Omega)$ using $\mathbf{w}_{0,\ldots,k}$. (eq.85)
$s = s + 1$
end while
$i = 0$
while $i < M$ **do**
$\mathbf{u}_i = \mathbf{u}_i / counting[i]$
$i = i + 1$
end while

Note that, following our notation, we have noted the subvectors of the vector solution u as $\mathbf{u}_i$. Another implementation aspect worth to be described is what concerns the *counting* vector. We know that the Ulam-Neumann algorithm allows us to compute the product between Ω and the solution vector u. A way of computing a single element of the solution $\mathbf{u}_i$ vector is setting the corresponding element of Ω as the identity matrix of size N, and the remaing elements of Ω to zero. Then, this can be repeated for all the elements of u. However, it is possible to implement that in a more suitable way: when we generate a Markov chain γ_k, we will devote it to compute an approximation of the element of the solution vector $\mathbf{u}_{\xi_0}$, where ξ_0 is the first element of the Markov chain. If we set $\mathbf{\Omega}_{\xi_0} = I_N$ and the remaining elements of Ω to zero, then $\Theta(\Omega)$ will be an estimation of the element ξ_0. This is the most straightforward way to employ an arbitrary Markov chain to find an element of the solution (to check this, take a look at equation 54). With the *counting* vector, we keep a record of how many times the algorithm has contributed to find an estimation of certain element of the solution, and then we use this information to compute the average.

In short, the *counting* vector strategy is just a way to abstract ourselves from Ω. Instead of imposing which element of the solution we are computing each time, we let the Markov chain 'decide' it. It would also be possible to force the first element of the Markov chain to coincide with the element of the solution we are trying to compute, however, our approach has an advantage: by choosing a right initial probability vector P_i, we could put more computational effort on computing the most important elements of the solution.

3.4 Hybrid Algorithms

The raw Ulam-Neumann algorithm is rarely used to solve a linear systems of equations, due to its rather slow convergence. Usually, it is employed to accelerate parts of deterministic algorithms where accuracy is not critical. For example, Halton [24] developed a sequential method that uses the Ulam-Neumann algorithm to accelerate the computation of a certain correction factor within every iteration of the classical scheme. Also, the works of Dimov and Alexandrov provide a way to accelerate the convergence of the Ulam-Neumann algorithm by reducing the norm the matrix. Another very interesting approach is the one provided by Alexandrov in works as [25], where the Monte Carlo method is used as a preconditioner by finding an approximate inverse of the system matrix. We are going describe and implement two of the methods [24] and [25].

Note that, in these cases, the purpose of the Ulam-Neumann algorithm is different from what we have previously described. Here we are not using a Monte Carlo algorithm for solving a system so large that it is unfeasible to access all its elements, but we are taking advantage of its speed for finding solutions that are still very large, but tractable.

3.4.1 The Halton Algorithm

The Halton algorithm is, in fact, a Richardson iteration accelerated by a correction, which is computed through the Ulam-Neumann algorithm. Here, the function BlockMC$(A, c) \approx A^{-1}c$ represents the blockwise Monte Carlo algorithm that we have described in section (alguna). Thus, the Preconditioned Sequential Monte Carlo algorithm would be:

ALGORITHM: Halton

Input: A, b, Monte Carlo execution parameters
Output: u_{SMC}

$\hat{A} = EA$ ▷ Preconditioned A
$\hat{b} = Eb$ ▷ Preconditioned b
while condition not met **do**
  $r_i = \hat{b} - \hat{A}x_i$
  $\delta x_i = \text{BlockMC}(\hat{A}, r_i)$ ▷ Monte Carlo approximation of $A^{-1}r_i$
  $u_{i+1} = u_i + \delta x_i$ ▷ Correction of the current solution
  $i = i + 1$
end while
$x_{SMC} = x_i$

Here, the Monte Carlo algorithm has been used to compute the correction δx_i, which is an approximation of $\hat{A}^{-1}r_i$. Note that this correction should always be computed in an approximated way, since computing it exactly would be equivalent to solving the whole system.

In our case, not only the Ulam-Neumann algorithm takes advantage of the matrix structure for efficiently multiplying matrix blocks, but also the product $\hat{A}x_i$ is executed in a blockwise case, where the multiplications are accelerated through FFT.

3.4.2 The Sparse Approximated Inverse Preconditioner (SPAI)

This algorithm, proposed by Alexandrov [25] employs the Ulam-Neumann series to precondition a linear system. Let us remember that the objective of a preconditioner is, in general, to reduce the condition number $\sigma(A)$ of matrix A, where $\sigma(A) := \max_i |\lambda_i| / \min_i |\lambda_i|$, where λ_i are the eigenvalues of A [Algun numerico que no sea quarteroni]. We would like the condition number to be as small as possible. The best possible preconditioner we can apply A would be A^{-1}. When A^{-1} is applied, the condition number acquires its minimum possible value 1, which means that the system is solved ($A^{-1}Ax = x$). Of course, it is not feasible to compute A^{-1}. However, rough approximations of A^{-1} can be suitable preconditioners. The SPAI algorithm employs the Ulam-Neumann series to find an approximation of A^{-1}. Then, this approximate inverse is used as a preconditioner.

The algorithm decomposes the matrix A as

$$A = A_d - C \tag{89}$$

where A_d is a diagonally dominant matrix and C is a diagonal matrix, whose elements c_i are selected depending on the norm of A in such a way that the diagonal dominance condition of A_d is fulfilled. For more details see [25].

Then, the Ulam-Neumann alghorithm is is used to compute an approximation of the matrix A_d. Since the matrix A_d is diagonally dominant, but does not necessarily fulfill the conditions for the Monte Carlo algorithm to converge, the preconditioning for dominant diagonal matrices should be applied to obtain a matrix $\hat{A}_d = EA_d$.

The Ulam-Neumann algorithm can be easily extended to find the inverse of a matrix. We solve the linear system $\hat{A}_d x_i = a_i$, where a_i is the i-th column of the matrix $\hat{A}_d$ and x_i the i-the column of the inverse matrix. Although this may seem extremely expensive, we should take into account that this is applied on sparse matrices where many of the elements are zero, or in dense matrices where many elements are almost zero. Consequently, and due to the fact that the transition matrix P favors computations on blocks with large

norm, many of the elements of x_i will actually never be computed, and will be left as zero.

Note that, in our case, this is also applied at the block level. So, if the matrix is divided into $M \times M$ blocks of size $N \times N$, the equation we are solving is $\hat{A}_d X_i = A_i$, where $\hat{A}_d$ is the system matrix, A_i is a column formed by matrix blocks (size $MN \times N$), and X_i is the corresponding column of blocks for the matrix $\hat{A}_d{}^{-1}$. The algorithm that solves an equation of these characteristics ($AX = B$), where X and B are block columns, while be noted as BlockBlockMC(,).

Finally, the matrix A^{-1} is recovered from the inverse of $\hat{A}_d$ by recursively applying the Sherman-Morrison formula [25], namely

$$B_k^{-1} = B_{k+1}^{-1} + \frac{B_{k+1}^{-1} S_{k+1} B_{k+1}^{-1}}{1 - trace(B_{k+1}^{-1} S_{k+1})} \tag{90}$$

ALGORITHM: SPAI

Input: A, Monte Carlo execution parameters
Output: A^{-1} (approx)

$A_d = A + C$ ▷ C is diagonal, A_d is diag. dominant
$\hat{A}_d = EA_d$
for all A_i (blocks level cols. of $\hat{A}_d$) **do**
  $X_i = \text{BlockBlockMC}(\hat{A}_d, A_i)$
  Set X_i as the i-th block level column of $\hat{A}_d{}^{-1}$
end for
$A^{-1} = \text{recursiveSM}(A_d^{-1})$ ▷ Recover A^{-1} by recursive Sherman-Morrison

---

Even with the Monte Carlo method, computing a preconditioning matrix is expensive, and this algorithm may be specially useful when the system is going to be solved for several right hand sides, since the preconditioning operation is common for all of them. In our physical case this is a very likely scenario, since it would correspond to computing how a certain dielectric array behaves under different electromangetic fields.

4 Numerical Examples

On this section we present a set of computational examples of what has been discused before. We will present convergence results of Halton and SPAI algorithms when the computations are accelerated thtough FFT, but also examples of other crucial numerical aspects are presented.

Also, we will present some considerations about the computational issues associated with this kind of problems.

4.1 Physical Meaning of Diagonal Dominance

As we have seen in section 3.3.3, the block level diagonal dominance of the system plays a fundamental role. We saw that the preconditioning (82) that preserves the Toeplitz structure fo the algorithm converges if the block level norm (83) is less than one. For convenience, we review this condition here:

$$\|H\|^{\text{Block}} = \max_i \Big(\frac{|\beta_i|}{d_i} + \|\mathbf{I}_M - \beta_i \frac{\mathbf{Z}_{ii}}{\|\mathbf{Z}_{ii}\|}\| \Big) < 1 \tag{91}$$

The constant d_i represents the block level dominance of the matrix, it is, how large is the norm of the diagonal blocks compared with the sum of the norms of blocks in the same block level row. In order for $\|H\|^{\text{Block}}$ to be smaller than one, we would like d_i to be as large as possible. Fortunately, this naturally happens in our systems. Let us focus, for example, in the physical system depicted in fig.4. It consist of an array formed by nine square patches of a size L of 100 nanometers. The electric permittivity of the material has been set $\varepsilon = -1 + i$, a classical plasmonic permittivity. The wavelength of the impinging field is of 400 nanometers. Each one of the square patches has been discretized into a grid of 10×10 elements.

In the example of (fig.4) the distance between the centers of the patches is $2L$. Let us examine the structure of the system matrix. The upper left plot of (fig.5) show the absolute value of the elements of the Z_{xx} matrix associated to the this system. We can easily distinguish the blocks of the matrix (9×9 blocks). The block Z_{xx}^{ij} represents the interaction between the patch i and the patch j. The 9 blocks in the diagonal are larger than the other blocks because the physical interaction between elements of the same patch are strong. However, the interaction between blocks that are far away are weak, and this implies that the elements of blocks away from the diagonal are smaller. Due to this, the matrix Z_{xx} certainly tends to show diagonal dominance in the block level.

If we increase the distance between the patches, the matrix will gain block diagonal dominance. This can bee seen in the remaining plots of (fig.5), that show what happens when the distance between centers is $2L$, $3L$, $10L$ and $20L$. The diagonal blocks do not change, since the distance between points in the same block remains the same. But the interactions between different blocks become weaker as the distance between them increases. So there more separated the blocks are, the more separated the patches are, the more the block level diagonal dominance increases.The values of minimum and

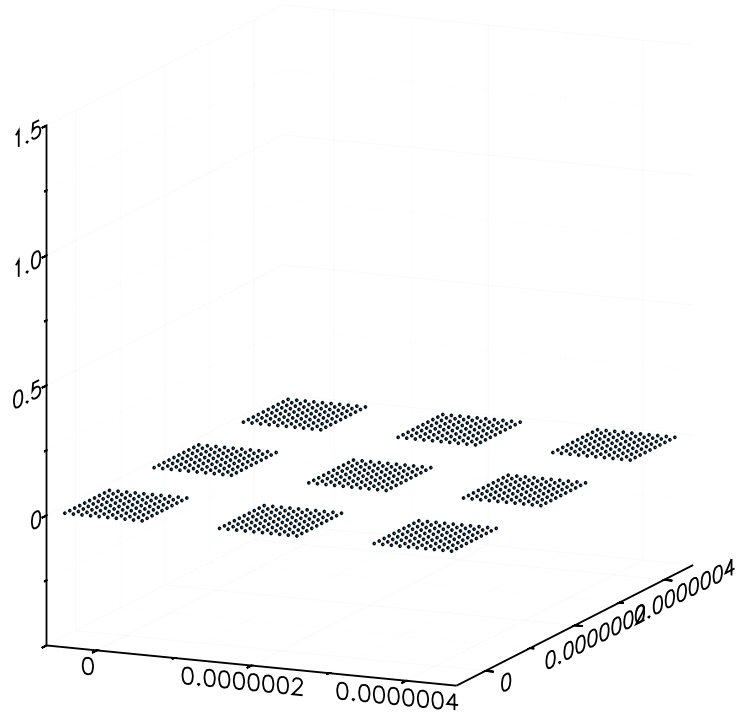


Figure 4: Nine array of 9 square patches. Each point represent an element of the discretized grid. For every element (point) there will be three unknowns correrresponding to the three components of the electric field in that point. Patch side is 100 nm, wavelength is 400 nm, and the permittivity is $\varepsilon = -1+i$

maximum d_i for each case are displayed on table 1 it is. We can see that, in all cases, the dominance factor is larger than one. This implies that the matrix is block level dominant diagonal.

Distance between patches	$\min_i d_i$	$\max_i d_i$
$2L$	10.15	15.73
$3L$	20.18	29.76
$10L$	92.82	131.13
$20L$	199.26	278.22

Table 1: Minimum and maximum dominance factors for the different physical cases, for the Z_{xx} matrix.

When the dominance factor is large enough, the convergence condition

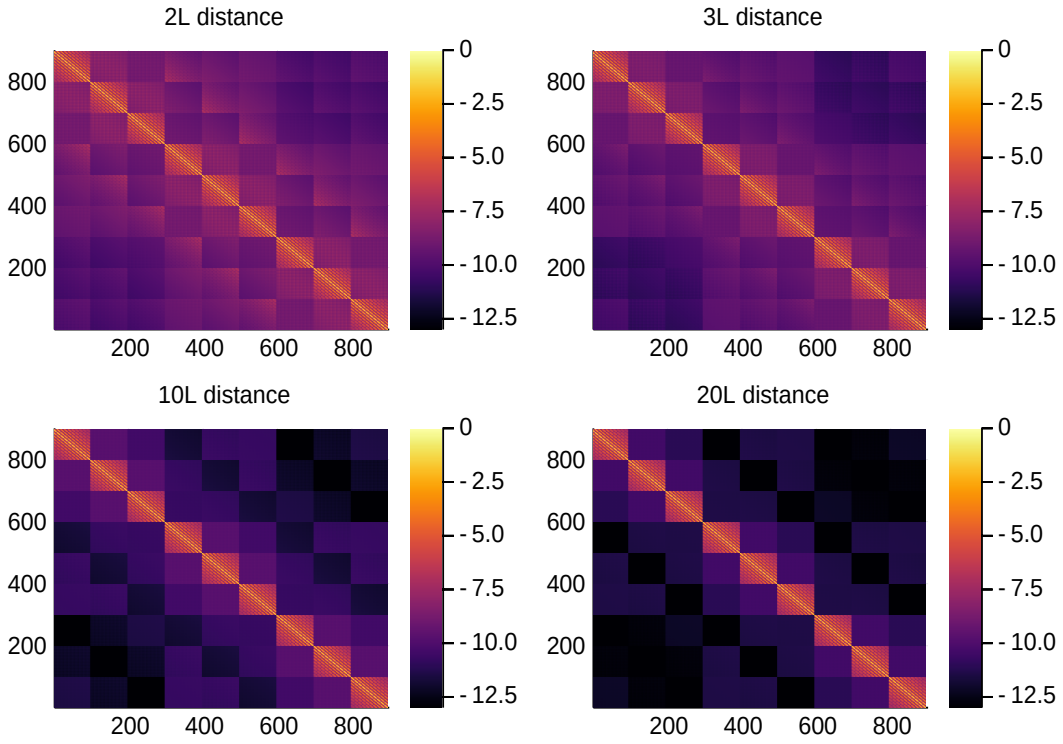


Figure 5: Comparison of the block Z_{xx} of the system matrix for the array presented in (fig.4). The absolute value of the elements of the matrix is displayed in a logarithmic scale. The different plots represent the cases where the patches are separated a distance of $2L$, $3L$, $10L$ and $20L$. We can easily distinguish the block representing the interaction between a patch and another. The blocks on the diagonal, that represent the interaction of a patch with itself, do not change when the distance between the patches increases. However, the interactions between different patches gets weak as the distance is increased.

can be written as

$$\|H\|^{\text{Block}} \approx \max_i \left(\|\mathbf{I}_M - \beta_i \frac{\mathbf{Z}_{ii}}{\|\mathbf{Z}_{ii}\|}\| \right) < 1 \tag{92}$$

Even in this case, finding a proper β_i is not automatic. But again, we benefit from the physical structure of the problem. First of all, let us remember that the block $\mathbf{Z}_{ii}$ is Toeplitz. Let us now imagine that it was also diagonal. Due to the Toeplitz structure, all the elements in the diagonal would be equal, and the matrix $\mathbf{Z}_{ii}$ would be written simple as $p_i\mathbf{I}$, where p_i is the value of the elements in the diagonal. In this case, it would be trivial to find a right β_i.

The matrix $\mathbf{Z}_{ii}$ is certainly not diagonal. However, the same physical argument we apply to explain the block level structure of Z_{xx} is applies withing the same block: elements in the diagonal or not far away describe interactions between elements that are physically close, and are strong, whereas elements away from the diagonal describe interactions between physically distant points, so they are small. As a result, the block $\mathbf{Z}_{ii}$ has a strong diagonal component, which makes easier to find a right β_i (it will be trivial in the perfectly diagonal case).

4.2 Impact of Block Size

The size of the block is another important factor to take into account. Although during this work we are assuming that we work with the blocks of the maximum possible size (the maximum blocks that have Toeplitz structure), it would be possible to take blocks of smaller size by subdividing these blocks (a subblock of a Toeplitz matrix is also Toeplitz).

The cost of a matrix vector product is, in general, $O(n^2)$. When the product is accelerated by FFT, the cost reduces to $O(n \log n)$. This implies that, as the size of the block increases, the benefit of using FFT gets more noticeable. In fact, even if mathematically the the cost of a FFT multiplication is never larger than the cost of a normal multiplication, it is possible than, computationally, the FFT is more expensive that the normal multiplication. This is due to the computational overheads associated to the FFT algorithm.

Again, we realize about the convenience of using large blocks.

4.3 Convergence of accelerated Sequential Monte Carlo

As we discussed before, the Sequential Monte Carlo (SMC) algorithm is a variation of the Richardson algorithm that employs the Ulam-Neumann method to find, at each iteration, a correction for the current solution of the problem. This correction is the result of solving a SLAE.

When the matrix is formed by blocks that have a Toeplitz structure, then we can accelerate the Ulam-Neumann algorithm by employing its blockwise version, accelerated through FFT. Since the most expensive part of every iteration of this algorithm is precisely to solve the SLAE in order to get the correction, then we can expect the improvement of the FFT acceleration to be very noticeable.

Figure 7 depicts how the accelerated version of the Sequential Monte Carlo converges faster due to the FFT acceleration. Take into account that the two cases in fig.7 are blockwise. The conventional Monte Carlo version converges much more slowly.

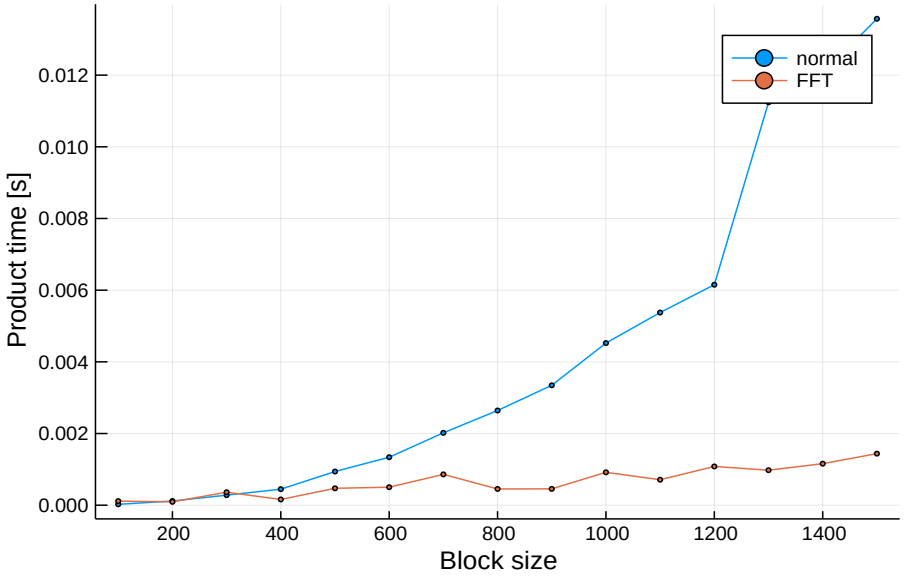


Figure 6: Time required for executing matrix vector products with FFT $O(n \log n)$ and with the regular n^2 algorithm.

4.4 Blockwise SPAI preconditioner

As we explained in the previous section, this SPAI preconditioner employs the Ulam-Neumann series to find an approximation of the inverse of the system matrix. This approximate inverse is used as a preconditioner.

Again, if the problem has the right block Toeplitz structure, we can use FFT to accelerate the computation of the matrix products. The improvement is depicted in fig.8, that shows the condition number of the system (thus the effect of preconditioning) depending on time. Of course, it would not have sense to precondition a system until its condition number is one, for that implies that the system is completeky solved. It is than in this case just to illustrate the behavior of the preconditioner. The fact that this matrix is much smaller than the used for the example of section 4.3 explains the apparent contradiction between the execution times in the examples of fig.7 and fig.8: it seems that in both cases the timings are similar, although on fig.7 we are solving a system and in the case of fig.8 we are inverting the whole matrix, which is much more expensive. We have used a relatively small matrix in the example of fig.8 because computing the condition number of the matrix involves finding its eigenvalues. This is a very expensive operation, thus, we have reduced the size of the matrix to be able to run the experiment.

We should point out that, after applying this preconditioning, the matrix

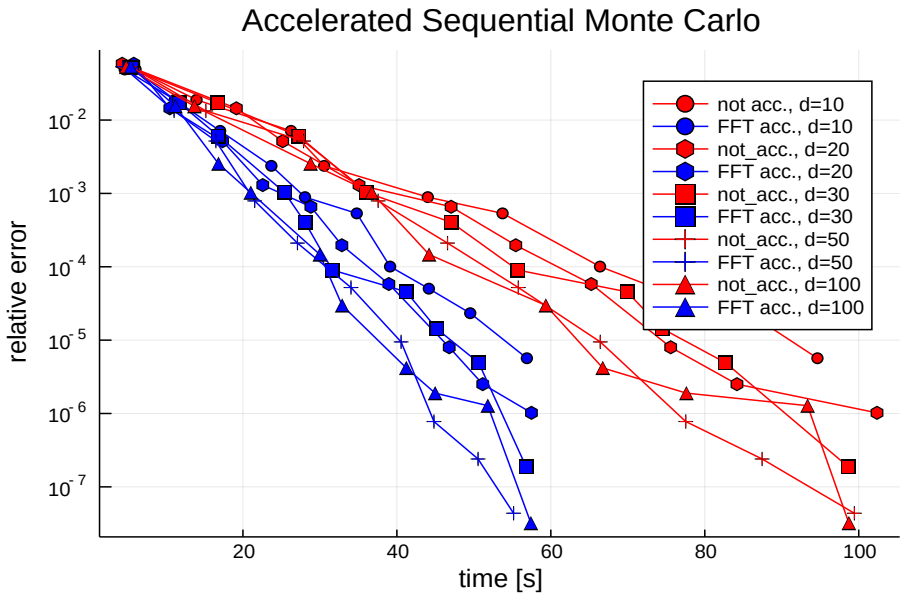


Figure 7: Comparison of the SMC algorithm executed in a blockwise way, without and with FFT acceleration. This corresponds to a system of 50×50 Toeplitz blocks, each one of size 800×800. We the Ulam-Neumann algorithm uses 10 chains with termination probability $T_i = 0.3 \, \forall i$ .The dominance factor is noted as d.

will loose its block Toeplitz structure, so the solver that we apply later would not be able to take advantage of the Toeplitz structure of the blocks in order to accelerate the computations.

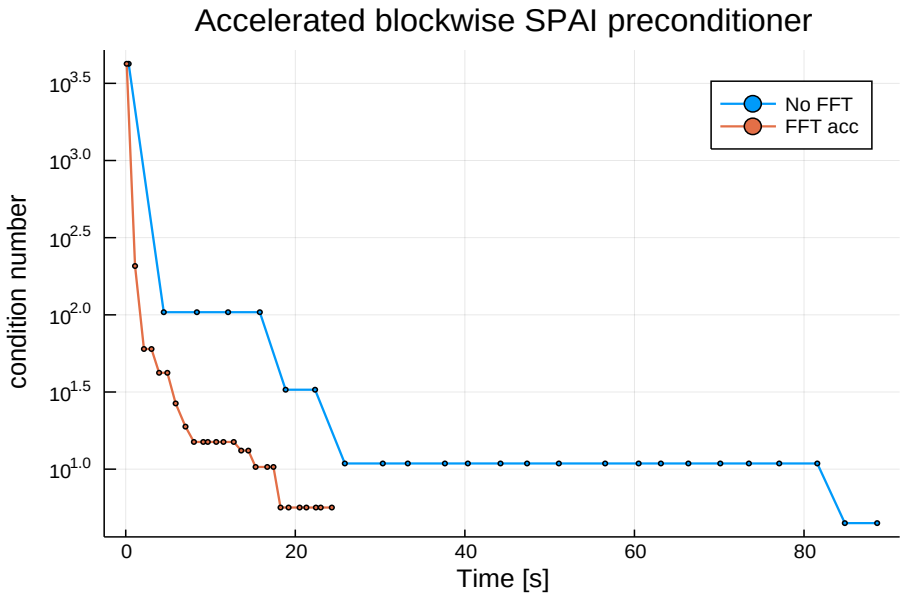


Figure 8: Comparison of accelerated and non accelerated SPAI preconditioners.

4.5 Computational issues

4.5.1 Scalability and Parallelism

Along this work we have highlighted the HPC vocation of our algorithm. However, up to this point, we have not provided clear evidence of that, only of the properties of the blockwise accelerated version of the Ulam-Neumann algorithm.

The answer to this relies on the Curtiss formula [3]. Curtiss used the Central Limit Theorem to provide an estimation of the complexity of the Ulam-Neumann Monte Carlo method. If we are using the Ulam-Neumann approach to compute a single element of the solution and our goal is to reduce the initial error by a given factor ξ and the iteration matrix of the Neumann series is γH with $0 < \gamma < 1$ (recall that this Neumann series corresponds to a stationary Richardson iteration), then the work required to fulfill the error objective with a 95% confidence interval is

$$WORK_{MC} = \frac{1}{1-\gamma}\left(1 + \frac{2}{\xi^2}\right) \tag{93}$$

whereas the work associated to the iterative method is

$$WORK_{Iter.} = \left(1 + \frac{\log \xi}{\log \gamma}\right) n^2 + n \tag{94}$$

where n is the number of elements on the solution vector.

It should be pointed that the complexity of the Monte Carlo approach does not depend on n. Thus, for small systems, the iterative method is more efficient than the Monte Carlo methods, whereas for very big systems, the Monte Carlo method is better. There exists a breakthrough point where n is such that $WORK_{MC} = WORK_{Iter.}$, up to which Monte Carlo is better.

Certainly, this is for computing a single element of the solution. If we are computing the whole solution vector (n elements), then the work associated to the Monte Carlo method has to be multiplied by n. The Monte Carlo method will lose then part of its advantages, but even in that case the iterative work grows as $O(n^2)$ and the Monte Carlo at $O(n)$. So, above a certain breakthrough n, the Monte Carlo method would be more convenient.

The Curtiss formula can be easily generalized to the blockwise case. Then, all the reasoning above also applies for our blockwise algorithm.

Regarding parallelism, the Ulam-Neumann method is intrinsically parallel. All the samples evaluated by the Monte Carlo algorithm can be evaluated in a completely independent way. Thus, the only communication needed will be the final reduce operation between the different processors. Note also that the division of work between the processors can be done in two different ways. One possibility is that every thread computes a single element of set of elements of the solution (thus a part of the vector) and then all the threads put their data in common. This can be done through forcing the first element of the Markov chain to take the value corresponding to the element we are computing. The other possibility is that all the threads compute an approximation of the whole vector, and then the results are conveniently averaged. Note that we can also think of hybrid approaches where, for instance, several threads are assigned to compute a specific set of elements of the solution, and finally they compute the average of their respective approximations. In this work, the we have parallelized the execution following the second approach among eight nodes.

Certainly, the systems we have studied are below the Curtiss breakthrough. They are not big enough for the Monte Carlo methods to be more suitable than the classical iterative solvers. Nevertheless, that does not nullify the validity of our reasonings. The theoretical proof of the scalability of the system could be stated as follows:

1. The Ulam-Neumann algorithm works more efficiently than iterative solvers for some cases. This is based on the Curtiss formula and many works in the Monte Carlo literature.

2. Our blockwise algorithm constitutes an extension of the Ulam-Neumann algorithm and inherits all its properties in a block level fashion. This is

specially useful when the blocks have a Toeplitz structure (as happens in certain problems arising from Computational Electromagnetics) in such a way that the products can be accelerated through FFT.

3. Then, for big enough cases of matrices formed by Toeplitz blocks, the Ulam-Neumann algorithm is more efficient that the corresponding iterative algorithm.

But were is the breakthrough point? We can make some rough estimation based on the Curtiss formula and our knowledge of the physical systems we are studying. We obtan a breakthrough point close to $n = 1000$. Since n stands for the number of side blocks and each block should have a size about 1000×1000 for taking profit of the FFT algorithm, then the breakthrough should be about one million of unknowns.

Also, its worth to point out that the system can be considered data parallel. The idea behind the Monte Carlo scheme is that we will access a small fraction of the elements of the matrix. Then, it is not likely that two threads will access the same element.

4.5.2 Recomputing blocks or accessing memory

One question that frequently appears on this kind of problems is whether is worth to compute the whole system matrix and access, when needed, its elements stored in memory, or if it is better to recompute the elements (blocks) when they are needed.

According to what we have said, the idea behind the application of a plain Monte Carlo method is that the system is so vast that we will not compute all its elements, but just take some representative samples. In that case, certainly, the correct approach is to compute the elements (blocks) only when we know for sure that we are going to need them, it is, when they have been randomly selected by the Markov chain. If the system is very large, it is even possible that we are not using elements (blocks) more than once.

However, when the Monte Carlo algorithm is used as a mean to accelerate some other method, as in the case of the Sequential Monte Carlo, then it is possible that we will need to compute the whole system anyway. In this case, the natural approach would be computing the whole system at the very beginning. We will point out again that in this case we are dealing with large systems, but not unmanageable as in the previous case.

4.5.3 Preconditioning issues

We have seen that, at some point, we need to precondition a diagonally dominant or block diagonally dominant system in order for its iteration matrix to converge under the Ulam-Neumann algorithm ($\hat{A} = EA$). This preconditioning involves computing the preconditioning matrix and performing a matrix-matrix multiplication. This can be very expensive.

In our experiments, we have applied the preconditioning in the most most straightforward, yet expensive, way, since our objective is just to perform a proof of concept of the blockwise algorithm. However, in a really applied case, some strategies should be considered in order to reduce the cost of the preconditioning operation.

For example, the case where the matrix has not been explicitly build and the blocks are computed on demand, the preconditioning will be applied dynamically to the blocks we need to compute. This will be the right approach when only a small fraction of the elements (blocks) are accessed during the execution of the algorithm.

Computing the preconditioning matrix requires knowing the norm of the blocks of the diagonal (A_{ii}). In general, since we are considering that the size of the blocks is small compared with the one of the whole matrix, this operation will be cheap. However, if we are interested into accelerating this process, and within the framework of the Monte Carlo approach, we can compute an approximation of the block norm by sampling, in a similar way as [26].

4.5.4 Computing the transition matrix

The transition matrix P assigns different weights to the elements or blocks of the system matrix. Be reminded that the element P_{ij} of the transition matrix represent the probability that, given an element of the Markov chain is the next element of the Markov chain is j, if the current element is i. A proper transition matrix will ensure that the Monte Carlo algorithm is picking elements that are relevant for finding the solution, whereas elements of lower weight are disregarded. Indeed, every transition matrix that fulfills the mathematical conditions (52) and (53) will work, however, if the transition probabilities do not take into account the weight of the elements of blocks, then the convergence may be prohibitively slow.

There are several different ways to compute P, but one of the most popular methods is the Monte Carlo Almost Optimal (MAO), that consists on setting the element P_{ij} as the absolute value of the corresponding element in the system matrix. As we have explained in section 3.3, we have used the

norm of the block ($\|A_{ij}\|_\infty$) and then, every row of the transition matrix is scaled to set the termination probability T_i to the desired value.

However, this would imply performing an expensive operation (finding the norm) in all the blocks of the matrix. There are several possible ways to overcome that difficulty.

One possibility is to follow an approach similar to the one proposed in the previous section, and sample the matrices to get an estimation of the norm. We can also build a synthetic transition matrix based on our knowledge of the structure of the matrix at the block level. In fact, such structure is very simple: as we can see in fig.5, our matrices show a very strong diagonal component. The norm of the blocks fades away as we separate from the diagonal. The Green's function (7) that describes the interaction between physical elements is $\sim 1/|r|$. The decay of block norm as we get away from the diagonal does not necessarily obey this relation, since the index j of $\mathbf{A}_{ij}$ does straightforwardly represent the physical distance between this block and $\mathbf{A}_{ii}$. However, as we said when discussing the diagonal dominance of these matrices, blocks away from the diagonal represent, as a rule of thumb, interaction between distant physical element, and we can take $1/|r|$ as an approximation of this decayment.

Thus, we can set the element P_{ii} of the transition matrix to an arbitrary constant c, and the other elements of the same line can be computed as $P_{ij} = c/|i-j|$. Finally, all the elements of the row are computed by another common constant, in such a way that the termination probability is the desired value (due to this adjustment, the value we chose for c is irrelevant).

5 Conclusions

We have proposed an extension of the Ulam-Neumann algorithm for solving systems of linear algebraic equations. The extension consists on applying the algorithm on the block level, instead of on single elements. Also, in order to analyze convergence of the block level Ulam-Neumann algorithm, we have extended to the block case a part of the work in [5]. In particular, we prove that, if the block level norm is smaller than one, it is trivial to find a transition matrix such that the algorithm converges. This algorithm is specially suitable for large scale parallel computing.

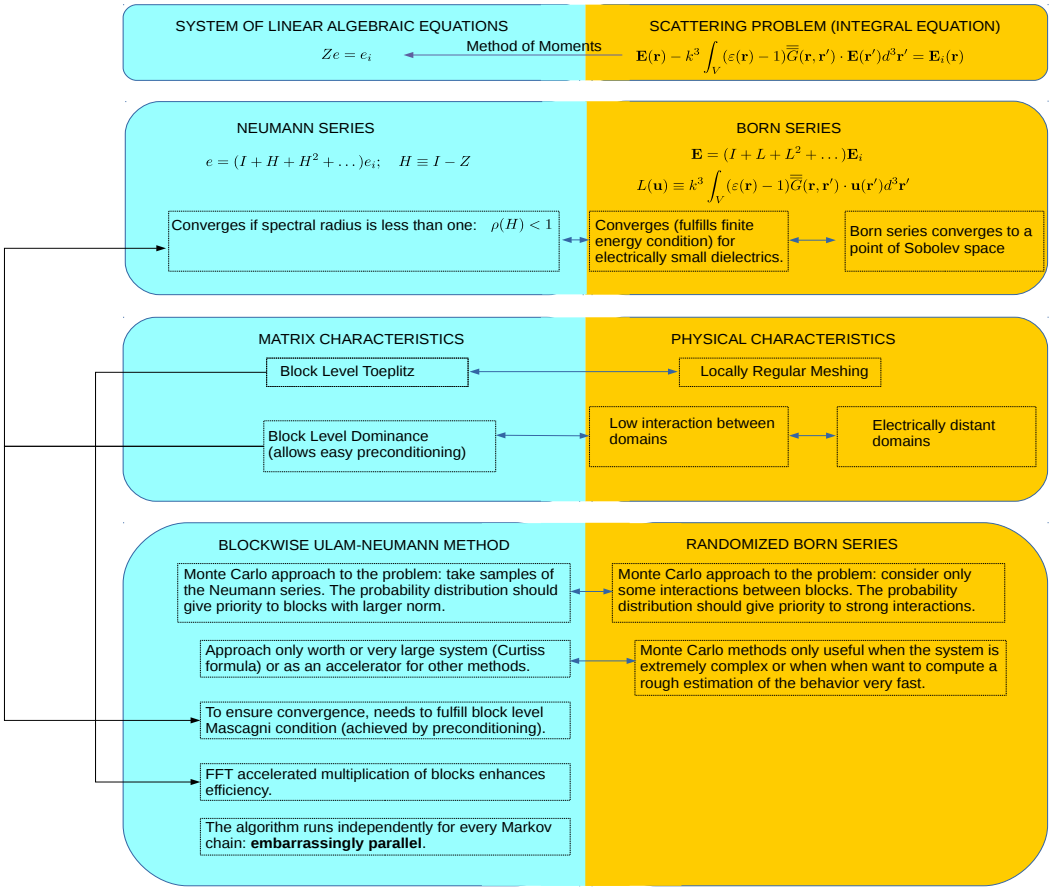


Figure 9: Map of the concepts involved in this work. Blue represents the computational side and orange the physical one.

The blockwise Ulam-Neumann algorithm is applied to a particular prob-

lem, namely the system of equations arising from the discretization of a scattering problem that appears in the context of photonics and nanophotonics. The blockwise Ulam-Neumann algorithm is specially useful for this problem due to the particular structure of the matrix. Due to the locally regular meshing, the matrix is formed by many blocks that have a Toeplitz structure. The multiplication of Toeplitz blocks in the blockwise Ulam-Neumann algorithm can be efficiently performed by FFT. Note that the advantage of the block structure would be lost with the classical Ulam-Neumann algorithm that considers single elements instead of blocks.

When the mathematical tool known as Neumann series is applied to the electromagnetic scattering problem it is known as Born series, due to historical reasons. This is not a mere historical remark, but provides deep insight in what we are doing. If the Neumann series is a Born series, and the Ulam-Neumann algorithm consists on randomly sampling a Neumann series, then, when applied to a electromagnetic scattering problem, the Ulam-Neumann algorithm can be considered a randomized Born series were samples are interactions between different points or domains of the physical system. Thanks to this connection, the existing knowledge about the Born series helps us on the application of our algorithm.

The Ulam-Neumann algorithm does not alway converge towards the solution. In fact, the convergence of the Neumann series is already problematic (converges when the spectral radius of the iteration matrix is less than one), and the convergence conditions for the Ulam-Neumann algorithm are even more restrictive. Fortunately, there exist some preconditioning methods that allow us to achieve convergence conditions. When the system is block level diagonally dominant, it is easy to find a preconditioning matrix such that the blockwise Ulam-Neumann algorithm will easily converge. And block level dominance is not a very strong requirement in our system: when the different domains interact weakly, the matrix becomes block level diagonally dominant. We can think of this in the following way: when the elements of the system are not tightly coupled, it is easier to compute approximations of the problem by sampling. Again this advantage comes when considering the matrix in a block level way. The matrix is block level diagonally dominant, but not diagonally dominant in the usual way. Thus, working in the block level also simplifies the task of finding a preconditioning such that the algorithm converges. In fact, if the matrix is not diagonally dominant in the block level, a possible solution would be considering larger blocks until the matrix fulfills the block level dominance condition. We have numerically checked that, in real physical cases, the block level dominance of the matrix allows the blockwise Ulam-Neumann algorithm to converge with the proper preconditioning.

The Ulam-Neumann algorithm is not usually applied directly. Sometimes, it is used in the framework of other algorithms to quickly find rough approximations of a SLAE. We have implemented two of these cases: the Sequential Monte Carlo algorithm and the Sparse Approximate Inverse Preconditioner based on the Ulam-Neumann method. In both cases the blockwise Ulam-Neumann algorithm converges, and the application of FFT to accelerate matrix-vector products allows us to speed up the algorithm.

Regarding large scale executions, the Ulam-Neumann algoirthm is suitable for parallel computing. This is mainly due to two reasons: first, because according to Curtiss formula, the algorithm is only better than the classical iterative solver for very large systems. This breakthrough point is certainly above the size that a desktop computer can manage. Second, because the algorithm is embarrassingly parallel. Our proof of concept shows that it is certainly parallel. Since there is no need for communication between computing threads, except for the final reduction operation, scalability should not be problematic.

To sum up, we have seen that the characteristics of the blockwise Ulam-Neumann algorithm and the ones of our physical problem complement each other in a very interesting synergy. Its randomized working principle is the suitable way for dealing with very large and complex physical problems. The fact that the algorithm considers matrix blocks instead of elements allows to take full advantage of the matrix structure by accelerating the multiplications through FFT. Besides, the diagonal dominant trend of the system matrix allows to easily find a preconditioner to ensure the convergence of the algorithm. And due to the inherently parallel nature of the algorithm, it can easily be employed to solve the huge systems arising from real world problems. For all these reasons the blockwise Ulam-Neumann Monte Carlo algorithm is a valuable computational tool for studying the behavior of complex photonic, nanophotonic and plasmonic materials.

References

[1] M. Hilbert and P. Lopez, "The Worlds Technological Capacity to Store, Communicate, and Compute Information", in *Science*, 332 (2011), pp. 6065.

[2] J. Curtiss, "A Theoretical Comparison of the Efficiencies of Two Classical Methods and a Monte Carlo Method for Computing One Component of the Solution of a Set of Linear Algebraic Equations", in *H. A. Mayer (ed.), Symposium on Monte Carlo Methods*, pp. 191-233. New york, NY: Wiley.

[3] A. Barto and M. Duff, "Monte carlo matrix inversion and reinforcement learning", pp. 687694. Morgan Kauf- mann, 1994.

[4] I. Dimov, *Monte Carlo Methods for Applied Scientists*, World Scientific Publishing, Singapore, 2008.

[5] H. Ji, M. Mascagni, and Y. Li, "Convergence Analysis of Markov Chain Monte Carlo Linear Solvers Using Ulamvon Neumann Algorithm", in , *SIAM J. Numer. Anal.,* 51 (2013) pp. 21072122.

[6] K. Sertel and J. Volakis, "Integral Equation Methods for Electromagnetics". 10.1049/SBEW045E, (2012).

[7] Novotny L, Hecht B, *Principles of nano-optics*, Cambridge Univ. Press, Cambridge, (2006).

[8] Gric, Tatjana, and Ortwin Hess. *Phenomena of Optical Metamaterials.* 2019.

[9] K. Inamdar, Y. P. Kosta and S. Patnaik, "Microwave Applications of Metamaterials Concepts," *2010 International Conference on Advances in Recent Technologies in Communication and Computing*, Kottayam, 2010, pp. 292-294. doi: 10.1109/ARTCom.2010.38

[10] A. Li, S. Singh, and D. Sievenpiper, "Metasurfaces and their applications, *Nanophotonics*, vol. 7, no. 6, pp. 9891011, 2018.

[11] S. A. Maier, *Plasmonics: Fundamentals and Applications.* New York, NY, USA: Springer-Verlag, 2007.

[12] J. van Bladel, *Electromagnetic Fields*, 2nd ed. Piscataway, NJ: IEEE Press, 2007.

[13] J. Rahola, "On the eigenvalues of the volume integral operator of electromagnetic scattering," *SIAM J. Sci. Comput.*, vol. 21, no. 5, pp. 17401754, 2000.

[14] Alexander Samokhin, Yury Shestopalov, and Kazuya Kobayashi. 2013. "Stationary iteration methods for solving 3D electromagnetic scattering problems". *Appl. Math. Comput.* 222 (October 2013), 107-122. DOI=http://dx.doi.org/10.1016/j.amc.2013.07.019

[15] M. F . Catedra, J . G . Cuevas and L. Nuno, A scheme to analyze conducting plates of resonant size using the conjugate gradient method and fast Fourier transform, *IEEE Trans. Antennas Propagat.*, vol. 36, pp. 1744-1752, Dec. 1988

[16] J.W. Cooley, P.A.W. Lewis, and P.D. Welch. *The Fast Fourier Transform and its applications*. IEEE Transactions on Education, 12(1):2734, 1969.

[17] J. W. Cooley, J.W. and Tukey, "An algorithm for the machine calculation of complex Fourier series", in *Math. Comp.*, 19, 297301, April 1965.

[18] Hanson, George W., and Alexander B. Yakovlev. *Operator Theory for Electromagnetics: An Introduction*. New York: Springer, 2002.

[19] Renardy, Michael, and Robert C. Rogers. *An Introduction to Partial Differential Equations*. New York: Springer-Verlag, 1993.

[20] Stein, Sherman K. *Calculus and Analytic Geometry*. New York: McGraw-Hill, 1973.

[21] A. Quarteroni, R. Sacco and F. Saleri, *Numerical Mathematics*, Springer-Verlag, Berlin, Germany, 2007.

[22] I. T. Dimov, T. T. Dimov, and T. V. Gurov, "A new iterative Monte Carlo Approach for Inverse Matrix Problem", in *J. Comput. Appl. Math.*, 92 (1998), pp. 1535.

[23] M. Benzi, T.M. Evans, S.P. Hamilton, M. Lupo Pasini, S.R. Slattery, "Analysis of Monte Carlo accel- erated iterative methods for sparse linear systems", in *Numer. Linear Algebra Appl.* (ISSN 1070-5325) 24 (3) (2017) e2088, https://doi .org /10 .1002 /nla .2088.

[24] J. H. Halton, "Sequential Monte Carlo techniques for the solution of linear systems", in *J. Sci. Comput.*, 9 (1994), pp. 213257.

[25] J. Strassburg and V. Alexandrov, “On Behaviour on Monte Carlo Sparse Approximate Inverse for Matrix Computations”, in *Proceedings of the ScalA 2013 Workshoop*. ACM, 2013, p. 6.

[26] A. Heldring, E. Ubeda and J. M. Rius, ”Stochastic Estimation of the Frobenius Norm in the ACA Convergence Criterion,” in *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 3, pp. 1155-1158, March 2015. doi: 10.1109/TAP.2014.2386306